DOCUMENT RESUME

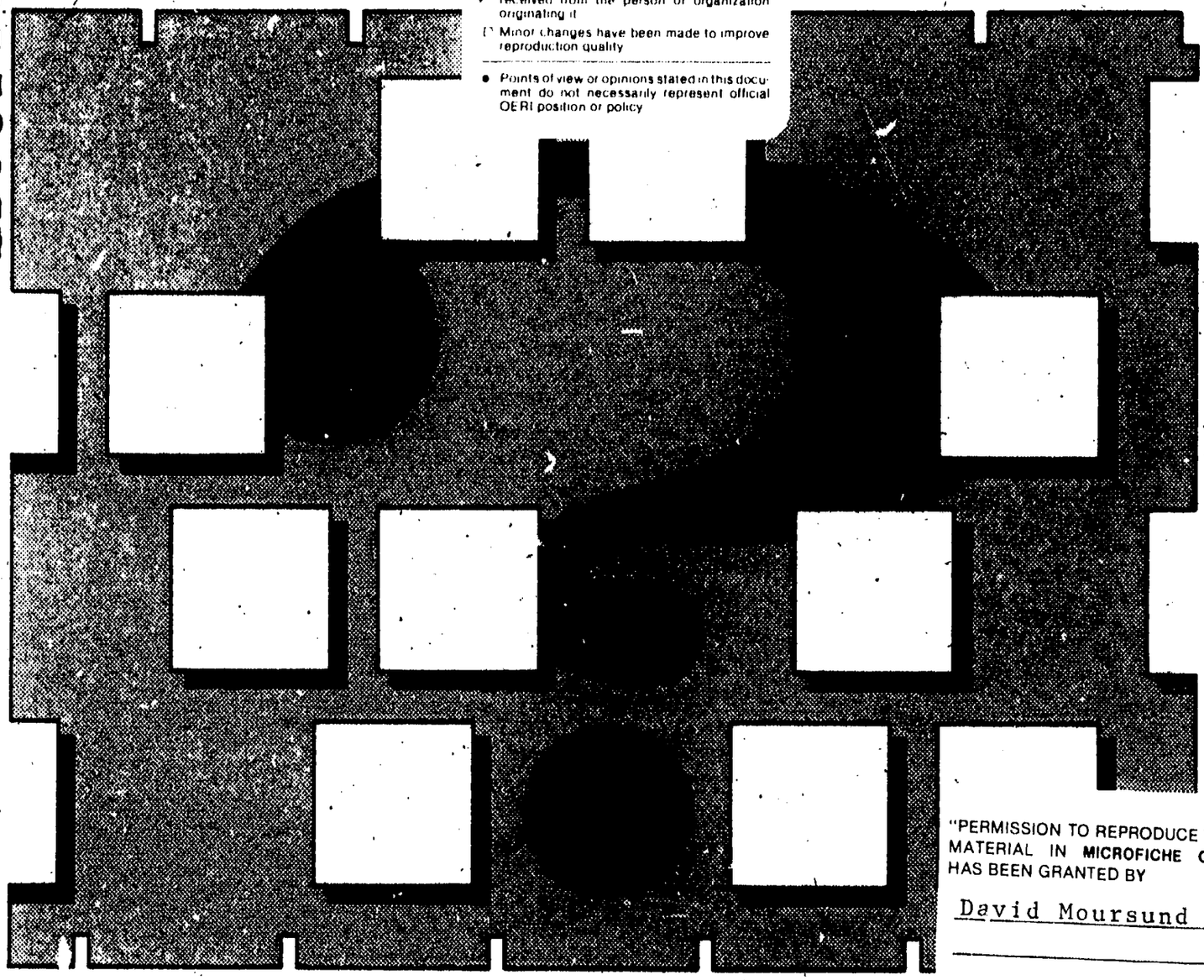
ED 331 468 IR 014 646

AUTHOR Yoder, Sharon; Moursund, David
TITLE Introduction to LogoWriter and Problem Solving for Educators.
INSTITUTION International Society for Technology in Education, Eugene, OR.
REPORT NO ISBN-0-924667-72-9
PUB DATE 90
NOTE 113p.
AVAILABLE FROM International Society for Technology in Education, 1787 Agate Street, Eugene, OR 97403-9905.
PUB TYPE Guides - Non-Classroom Use (055)

EDRS PRICE MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS *Computer Assisted Instruction; *Computer Software; Elementary Secondary Education; *Inservice Teacher Education; Microcomputers; Postsecondary Education; *Preservice Teacher Education; *Problem Solving; *Programing; Programing Languages; Word Processing
IDENTIFIERS *LOGO Programing Language

ABSTRACT

This book about Logo programming and problem solving is designed to introduce preservice and inservice teachers to problem solving in a Logo programming environment. Such a unit of study can be an important part of an introductory computers in education course for educators. Although Logowriter--a version of Logo--was developed by Logo Computer Systems, Inc., primarily for use on the Apple II, MS DOS (IBM compatible), and Commodore microcomputers, no specific computer hardware or version of Logo is required to use the ideas presented in this book. The following topics are discussed: (1) getting started with Logowriter; (2) using REPEAT and turtle move mode; (3) color and RANDOM, shapes and STAMP, FILL and SHADE; (4) mixing text and graphics; (5) writing procedures and more than one procedure; (6) designing programs; and (7) music. Appendices include a description of Logowriter keys, keyboard stickers, and a list of quick word references. (34 references) (DB)



LogoWriter for Educators: A Problem Solving Approach

ISTE Publications

By Sharon Yoder and Dave Moursund

About the Authors

Dave Moursund has been teaching and writing in the field of computers in education since 1963. He is a professor at the University of Oregon in the College of Education. There he teaches in and helps direct both a master's degree program and a doctoral program in computers in education.

Some of Dr. Moursund's major accomplishments include:

- Author or co-author of about 25 books and numerous articles.
- Chairman of the Department of Computer Science, University of Oregon, 1969-1975.
- Chairman of the Association for Computing Machinery's Elementary and Secondary School Subcommittee, 1978 - 1982.
- Founder, International Council for Computers in Education, (ICCE) 1979. The name of this organization was changed to International Society for Technology in Education (ISTE) in 1989 when it was merged with the International Association for Computing in Education.
- Chief Executive Officer, ICCE, 1979-1989.
- Executive Officer, ISTE, 1989-present.

Sharon Yoder has taught mathematics and computer science at the junior high and high school level for 15 years. Her most recent public school experience was as a secondary computer science teacher and a computer coordinator involved in developing system-wide computer curriculum and in planning teacher inservice training.

In addition, she has taught mathematics, computer science, and computer education at a number of universities in northeastern Ohio, including Kent State University, the University of Akron, and Cleveland State University. She has worked closely with the College of Education of Cleveland State University in developing their computer programming courses for teachers.

After a year as an Education Specialist for Logo Computer Systems, Inc., Sharon has returned to teaching. She is currently at the University of Oregon, where she teachers computer education courses.

For the past several years, she has conducted workshops and presented papers at conferences nation-wide, and has been involved in a number of book publishing projects, including the *Nudges* series. In addition, she has been a frequent contributor to *The Computing Teacher* and the *Logo Exchange*. She is currently editor of the *Logo Exchange* and a column editor for The Logo Center in *The Computing Teacher*.

Project Editors: Anita Best, Ellen Siegel, Neal Strudler
Cover Design: Percy Franklin
Production: Tamara Kidd

For Ordering and a complete catalog contact:

INTERNATIONAL SOCIETY FOR
TECHNOLOGY IN EDUCATION
University of Oregon
1787 Agate Street
Eugene, Oregon 97403-9905
503/346-4414
CompuServe: 70014,2117 BITNET: ISTE@Oregon

ISBN 0-924667-72-9

Introduction to LogoWriter and Problem Solving for Educators

Sharon Yoder

Dave Moursund

International Society for Technology in Education
1787 Agate Street
Eugene, Oregon 97403-9905

Contents

Preface

The Logo programming language was designed for children. Logo can be used to help create a fun, exciting, stimulating learning environment. Children of all ages, both youngsters and oldsters, enjoy working with Logo. Moreover, a Logo environment gives a window into the world of computers, computer programming, and problem solving using computers.

This short book is about Logo programming and problem solving. It is designed to introduce preservice and inservice teachers to problem solving in a Logo programming environment. Such a unit of study can be an important part of an introductory computers in education course for educators.

Computers and computer-related technology are having a profound impact on our society and have the potential to strongly affect education. There is general agreement that all teachers should have a functional level of knowledge of the roles computers play in the teaching and learning process. Many teacher training institutions now require all preservice teachers to take a course on computers in education. Many school districts are working to help all of their inservice teachers acquire a functional level of computer knowledge and skills.

The overall field of instructional use of computers can be divided into three main parts:

1. Computer science, computer programming, and the associated underlying theories.

2. Computer assisted instruction, computer managed instruction, and other aspects of using a computer system to help teach.

3. Productivity tools for use by students and teachers, such as a word processor, database, or gradebook program.

These three major categories are not distinct. A single piece of computer software or a single computer idea may fall into all three categories. Today's versions of Logo, as well as problem solving as a computer-related idea, each fall into all three categories. However, Logo falls most strongly into the computer science/computer programming category. This book focuses mainly on programming in Logo and those aspects of problem solving that fit well into discussions of Logo programming and of problem solving in a Logo environment.

Logo is a computer programming language that was specifically designed for use in schools. When Logo was first developed in the late 1960s, there were no microcomputers. The early versions of Logo contained none of the graphics capabilities common to all versions of Logo today. Interactive computer graphics was very expensive and had not yet come into common use. Computer use in elementary schools was mainly limited to drill and practice programs that focused on helping students to acquire basic skills. Thus, Logo was a revolutionary idea.

Over the years, Logo has changed immensely. Undoubtedly the greatest breakthrough was the late 1970s implementation of a graphics-oriented version of Logo that could run on an inexpensive microcomputer. Since then a variety of versions of Logo have been developed for microcomputers and have been widely distributed. Logo has been used with millions of students at all levels, ranging from preschool to graduate school. Logo has been the focus of hundreds of research studies. Seymour Papert's book (1980) has been a best seller for many years. In total, Logo has had and continues to have a major impact on the field of instructional use of computers.

This book provides a brief introduction to one particular version of Logo, called LogoWriter. This version of Logo was developed by Logo Computer Systems, Incorporated (LCSI). It is available for the Apple II series of computers, MS DOS (IBM compatible) computers, and Commodore computers. However, most of the ideas presented in this book are independent of any particular computer hardware or version of Logo.

This book can be used in a short, stand-alone course to introduce educators to some of the key ideas of Logo and problem solving. Alternatively, the material in this book might be covered in a two to three week part of a three credit course covering the fundamentals of computers in education. Such a course should have problem solving as a central and unifying theme. Keep in mind that the main reason computers are used throughout the world of business, industry, and government is that they are a very useful aid to problem solving. They are an aid to the human mind. Computers are a powerful mind tool.

Computers are important in education because they are a unique and powerful new aid to problem solving in every academic field. The literature on problem solving and the literature on Logo are both quite large. This short book does not attempt to be comprehensive in either field. Rather, it is intended to introduce a few of the underlying and unifying ideas in these fields, and to provide a methodology for studying them. This book draws heavily from previous works of the authors. Educators interested in a more comprehensive introduction to LogoWriter may want to read Yoder (1990), and those wanting a more comprehensive introduction to computers and problem solving may want to read Moursund (1990). A number of other sources of information are given in the References section. Readers seeking a continuing source of new ideas on the use of Logo in education should consider subscribing to the *Logo Exchange*, a periodical published by the International Society for Technology in Education.

Sharon Yoder and Dave Moursund
July 1990

Chapter 1
Computers and Computer Programming

Did you read the Preface? If you didn't, please go back and browse through it. The Preface is like an advance organizer. It helps orient you to thinking along the same lines as the authors are thinking.

A computer is a machine designed to rapidly and automatically carry out a detailed, step-by-step set of instructions. Such a step-by-step set of instructions is called a computer program. People write computer programs to help themselves and others solve certain types of problems.

The ideas involved in using the computer to write programs to solve particular kinds of problems are a central part of the objectives of this book. These objectives are:

1. To increase your understanding of the capabilities and limitations of a computer as an aid to solving problems.

2. To give you a brief introduction to the topic of "computer programming" and the process of writing computer programs to solve problems.

3. To help you learn the rudiments of writing computer programs in LogoWriter, a specific version of the Logo programming language.

However, these are actually secondary objectives. Logo was designed to bring a new dimension to education. The overriding objective of this book is to introduce you to a unique computer-based learning environment and to give you opportunities to practice working in this environment.

This book assumes that you, the reader, are a person who plans to become a teacher or is already a teacher. The book will help you to learn some key ideas about problem solving, computer programming, and the Logo programming language. Many of the ideas will be useful to you even when you are not working with computers.

The assumption is that many of the ideas we will cover will be relatively new to you. Thus you will have the opportunity both to learn some new ideas and to practice learning to learn. Learning to learn, and learning more specifically about how *you* learn, is fun, exciting, and critical to being a successful life-long learner.

In recent years there has been a lot of research on the value of learners keeping a journal and writing in the journal as they study a subject (Specht, 1990). As you use this book, you will want to introspect. You will want to carefully examine your own learning processes. You will want to keep a record of this introspection process in a journal. In the future, if you use Logo when working with children, you will find it helpful and rewarding to have them keep a journal.

Learning and Knowing

Have you ever asked yourself what it means to "know" something or to have learned something? Think of some small part of an academic area you know quite well. How do you know that you know it? How can someone else tell that you know it?

One answer is that you can make use of the knowledge. You can demonstrate to yourself and others that you can use the knowledge for personal purposes, accomplishing specific tasks, for helping other people, and so on. *Learning* and *doing*, or making use of one's knowledge, are closely intertwined. Indeed, we know that most students learn best by *doing*. For them, a good instructional environment is one that includes substantial opportunity to practice using what they are learning.

However, an equally important aspect of a good learning environment is feedback. The feedback can come from the learner. For example, consider a student who has written a poem. The student might say, "I really feel good about this poem. When I read it to myself, it brings a smile to my face. However, the rhyme scheme still doesn't sound quite right, and I want to say more about the cat."

Of course, the student can also get feedback from a teacher or from fellow classmates. Such external feedback is often very important. A teacher needs to be skilled in providing appropriate feedback to students.

Feedback While Doing Computer Programming

A computer environment adds an important new dimension to learning. A computer can help provide feedback to the learner. This book will help you to learn to write Logo programs. As you learn to write programs, you will get feedback from yourself, from other people, and from the computer you are programming. Programming means writing instructions to tell a computer what to do. As you attempt to write a program, you will have in mind what task you want the computer to accomplish. You will observe what the computer actually does (and/or fails to do). You and the computer together will provide feedback to yourself.

Sometimes this feedback will be positive and very rewarding, for example, "I wanted to tell the computer how to draw a picture of a house with smoke coming out of a chimney. My program works just exactly like I wanted it to."

At other times the feedback will be mixed or negative. "I wanted to have the computer draw a plane flying in a cloudy sky. Right now my plane looks more like a car and my clouds seems to be sitting on the ground. I'm making progress, but it's clear I've still got some problems with my program."

Errors (Bugs) in Computer Programs

Computer programmers call an error in a computer program a *bug*. This goes back to a time when a computer failed to function properly because a real insect (a "bug") got stuck in the circuitry. The process of removing errors from a program (removing bugs) is called *debugging*.

The metaphor of finding bugs and doing debugging, or detecting and correcting errors in one's work, is applicable both in computer programming and in many other endeavors. It is an important metaphor applicable in all problem solving tasks. It is very easy to learn and practice this metaphor when programming a computer, and this is one of the unique aspects of the computer programming environment. However, it is important to ask if the knowledge and skills you gain about debugging in a computer programming environment transfer to other areas.

Transfer of Learning

Very few people make a living writing Logo programs. For most people, the main reason for learning to write Logo programs is to gain knowledge, skills, and attitudes that transfer to other

situations. For example, you might become quite skilled at detecting and correcting bugs in your Logo programs. Does this help you to get better at detecting and correcting errors in some other activity that you do? The research suggests "Maybe."

Research on problem solving indicates that self-confidence and self-esteem in problem solving transfer among different problem solving domains (subject areas). For example, suppose that you are successful in learning to write Logo programs and feel good about your ability to do so. Quite likely this will help you as you work to learn another programming language. Seymour Papert (1980), who is considered to be the father of Logo, argues that there is considerable transfer of self-confidence and self-esteem from learning Logo to learning math.

Early researchers into problem solving in a computer programming environment were confident that they would find a great deal of transfer from this domain to other problem solving domains. They expected that the types of thinking needed to write computer programs would readily transfer to solving problems in science, mathematics, and the social sciences. They were disappointed. Teaching a child to write computer programs does not automatically produce significant gains in problem solving in other fields of endeavor.

There has been a great deal of research on transfer of learning. Here are a few things we know:

1. In *near transfer*, a person automatically (with little or no conscious thought) uses knowledge and skills gained in one situation to help solve a problem that occurs in a different situation. Every student is capable of doing near transfer. However, what is near transfer for one student may not be near transfer for another student. The nearness or farness of transfer is mainly dependent on the student rather than on what is being transferred.

2. The amount of transfer and the farness of transfer that occur can be increased by careful introspection, talking and thinking about transfer, and attempting to make transfers. Thus, a teacher can "teach for transfer" and a learner can consciously and actively "learn for transfer."

Every teacher and every student should have a good, conscious understanding of the latter point. Think of yourself as a learner, studying Logo programming and problem solving in a computer programming environment. The nature and extent that transfer occurs for you is dependent mainly on you. It will occur if you carefully and consciously work to help it occur. Similarly, if you teach Logo to your students, transfer will be substantially increased if you help your students work towards learning to transfer the ideas they learn. Each chapter of the book contains a specific discussion on problem solving designed to help increase transfer.

Summary

This book provides a brief introduction to writing Logo programs to solve problems. The overriding goal is to help provide an environment in which you can practice learning to learn. Secondary objectives include learning more about computers, problem solving, and the process of writing computer programs to solve problems.

Chapter 2
Keeping a Journal

Throughout this book, you will be encouraged to think about your own thinking, to *metacognate*. As indicated earlier, the key focus of your work with Logo is to give you experience in a particular learning environment: one in which you can explore, experiment, and receive feedback on an ongoing basis. As you work with Logo, you will have moments of frustration and moments of success. A journal is an excellent way to record your learning process and to keep a written trail as you think about your own thinking.

You should get a separate notebook that you always keep with you as you work at the computer or think about Logo programming. In it you should record your thoughts and feelings. Since a journal is a personal document, it can be quite informal. Your journal will become a record of your learning. It will help you document your progress. Further, it will help you see your successes at moments when you are feeling like a failure.

Here are some examples of typical journal entries.

"Today I started working with Logo. Much to my surprise, I was able to move the turtle quite easily."

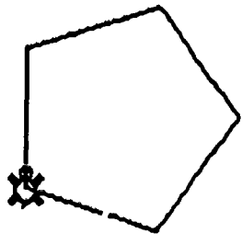
"This is the third time I have tried to figure out how to draw a triangle. I am VERY frustrated..."

"What a wonderful insight I had...I understand my own thinking so much better..."

"Logo really makes me think. The debugging process, while frustrating, forces me to examine my own reasoning."

"I intended to draw this:

...but it came out like this:



What happened?"

"At last I understand how procedures work....I think!"

"I'm beginning to understand Logo error messages. They really do provide useful feedback."

"I thought you had to be a brain to write computer programs. I'm doing really well, and I'm proud of myself."

"Mumble...mumble...mumble...dumb computers...dumb programming language."

Start each session entry with a date and time. Put down what you are thinking and take some time at the end of each session to reflect on what you have written and what you are thinking and learning. Make a conscious effort to tie in your work with Logo with the ideas you are learning about problem solving.

There are some suggested exercises and activities at the end of each chapter of this book. Some of them will ask you to make use of your journal. As you work on other activities, you will no doubt want to use your journal to document your progress.

Thinking and Metacognition

The human brain is very complex and contains a huge number of neurons. Think of a neuron as a long, very thin cell that may be closely interconnected with thousands of other neurons. When you are learning, you are building and strengthening interconnections among neurons. When you are thinking, many millions of neurons may be engaged. You do not consciously control all these neurons. Almost all your thinking activity occurs at a subconscious level.

However, by conscious effort, you can direct your thinking toward a specific topic or problem. Such thinking may seem like talking silently to yourself, creating pictures in your head, or creating sounds, smells, and feelings in your head.

Research on thinking indicates that we can all get better at thinking through study and practice. One way to do this is through metacognition–thinking about thinking. Another way is by carefully monitoring the outcomes of our own thinking processes. Become consciously aware of your more effective thinking processes and your less effective thinking processes. Then use this increased understanding of yourself to get to be a better thinker.

Research on metacognition indicates that even primary school students can learn to monitor their own thinking efforts and can use this activity to improve their thinking skills. Thus, teachers who work with Logo in elementary schools should be encouraged to have their students do metacognition. In a Logo environment, journaling can be a useful aid to metacognition. Research has indicated that journaling can be effectively used with children even at the primary school level (Specht, 1990).

Activities

Before you actually start on Logo programming, you may want to capture a snapshot of some of your current feelings and understanding about computer programming and problem solving in your journal. Here are some possible topics you might want to address.

1. What do I already know about computer programming, and what new things do I expect to learn?

2. What is "problem solving" and what do I really know about problem solving?
3. Will studying Logo and problem solving help me to be a better teacher?
4. How do I *really* feel about the idea of learning to write computer programs?
5. What are examples of some types of problems that computers cannot solve?

Chapter 3
Getting Started With LogoWriter

Starting LogoWriter

The easiest way to get started using LogoWriter is to have someone show you how to get started. The details of what to do at the beginning vary with the computer system and version of LogoWriter you are using. If you are using a computer system with a floppy disk drive then you will need two disks to begin:

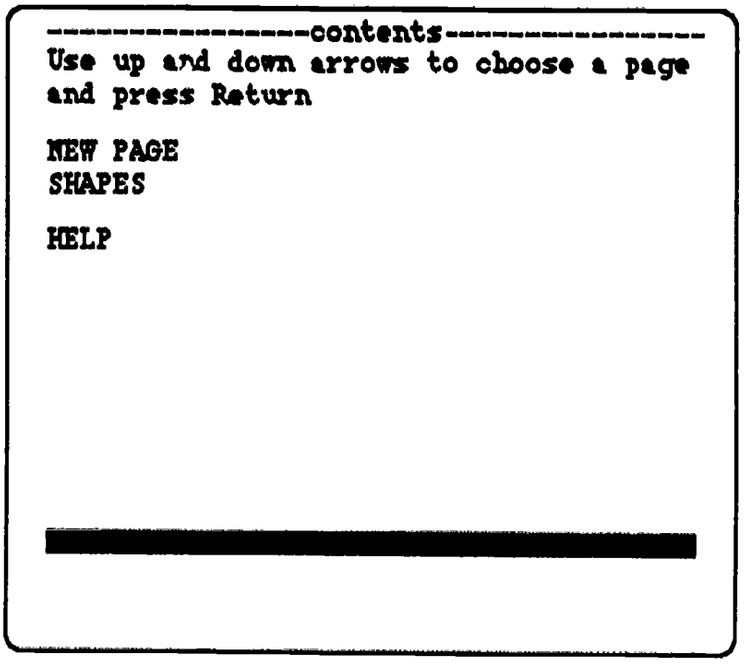
- A disk containing the LogoWriter language.
- A LogoWriter Scrapbook disk (this is a "files" disk).

It is assumed that your course instructor will provide you with these disks. (If you are using a networked LogoWriter system, the LogoWriter language will be a file on a hard disk rather than on a separate floppy disk. Other details about getting started will be different than what is described in these next few paragraphs.)

Begin by putting the disk containing the LogoWriter language in the disk drive of the computer and then start the system. The specific keys to press depend on the kind of computer you are using. It is assumed that your course instructor has provided you with instructions for the specific hardware and version of LogoWriter you are using.

The disk drive will whir and buzz, and then you will see a screen containing the word "LogoWriter" written in large letters.

Take out the disk containing the LogoWriter language and put it away. It is needed only when you are starting LogoWriter. Put in your *Scrapbook disk*, and press the *Return/Enter key*. You then see the Contents page, which shows a list of pages (files) that are on your disk.

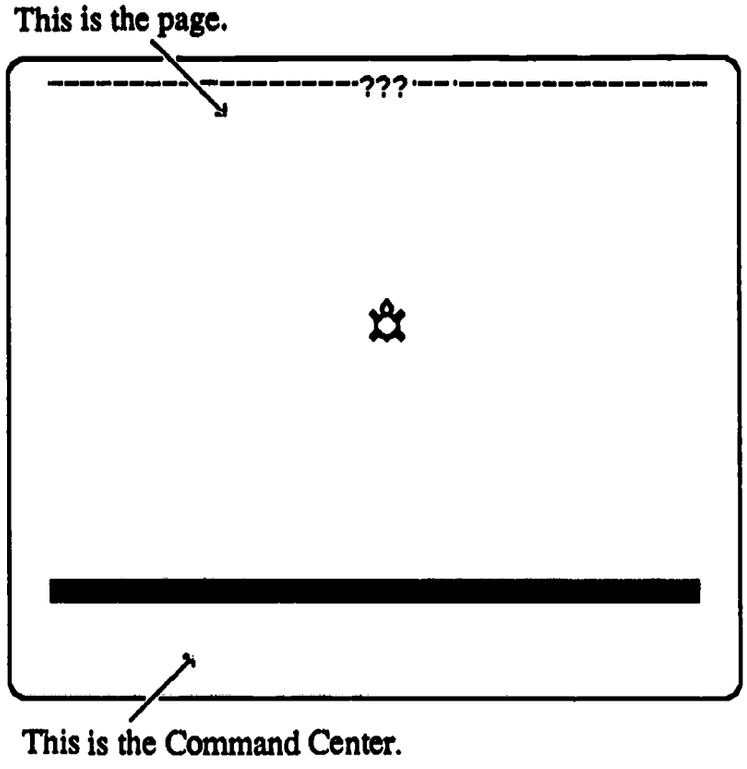


To select a page from the Scrapbook disk, use the arrow keys to move the small square, called the cursor. When the cursor is on the page name you wish to use, press Return/Enter. To get started, select NEW PAGE and press Return/Enter.

Note: LogoWriter comes with "keyboard stickers" to help you remember what keys to use for various activities. You can find copies of these stickers along with a list of the various key strokes that are used with several different versions of LogoWriter in the Appendix of this book.

It is assumed that as you read the sections of this book, you will have access to a computer so that you can try out new ideas as they are presented. You will understand the material much better if you become actively involved. At the same time, you should have your journal close at hand. You can note discoveries you make as you explore. You can jot down questions that arise so that you can seek answers at a later time.

If you started LogoWriter correctly and selected a new page, then your screen should now look like this:



This is a blank LogoWriter screen for you to work with. The area above the dark line is called the Page. The area below the dark line is called the Command Center. You will use the Turtle to draw.

Whenever you get a new page, it is a good idea to give it a name. To do this, you must type

```
NAMEPAGE "name.you.want (Return/Enter)
```

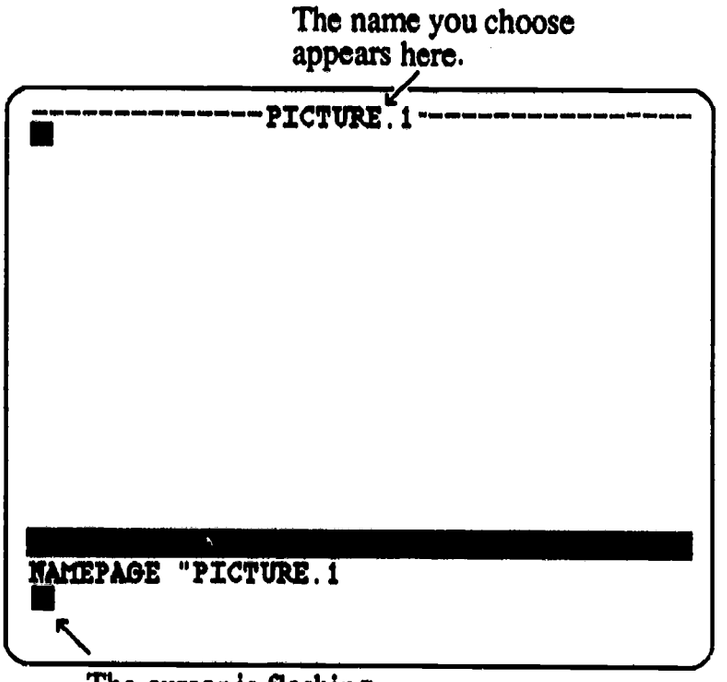
or use the abbreviated form

```
NP "name.you.want (Return/Enter)
```

in the Command Center. Note that *name.you.want* is the word you want to use to name your page. Be sure to put a space after NAMEPAGE. Put a " (quotation mark) before the name. Do not leave a space after the quotation mark.

For example, you might type

```
NP "PICTURE.1 (Return/Enter)
```



If you select a name that is more than one word long, use a period between the words. Do not leave blank spaces between the words. You can name your page almost anything you wish. LogoWriter displays a message telling you if you use illegal characters or if your name is too long. The name you type appears at the top of the page in place of ???.

If you make a typing error and detect it before you do a Return/Enter, you can correct it by using the Delete/Backspace key. Often when you make a mistake in typing, Logo will respond with a message such as

```
I don't know how to ...
```

Simply try again, paying particular attention to the spaces. The "I don't know how to ..." is an that you will see frequently, for example when you make a keyboarding error. Right now would be a good time to deliberately make an error so that you begin to get used to dealing with this message. (Does the idea of deliberately making an error bother you? If so, write about these feelings in your journal.)

Like any other programming language, Logo has a number of specific rules that you must learn in order to use the language well. At first many of these rules will seem arbitrary. Why, for example, must there be only one quotation mark before a page name? Why must a page name be a single word with no blank spaces in it? As you learn more about the structure of the Logo language, you will begin to see reasons for these rules.

Drawing With the Turtle

Now you are ready to start drawing with the turtle. The turtle holds a pen that draws lines as the turtle moves. Try typing

```
FORWARD 50 (Return/Enter)
```

or use the abbreviated form

```
FD 50 (Return/Enter)
```

Be sure you type a zero (0), not the letter O ("oh"). The turtle moves forward 50 "turtle steps."

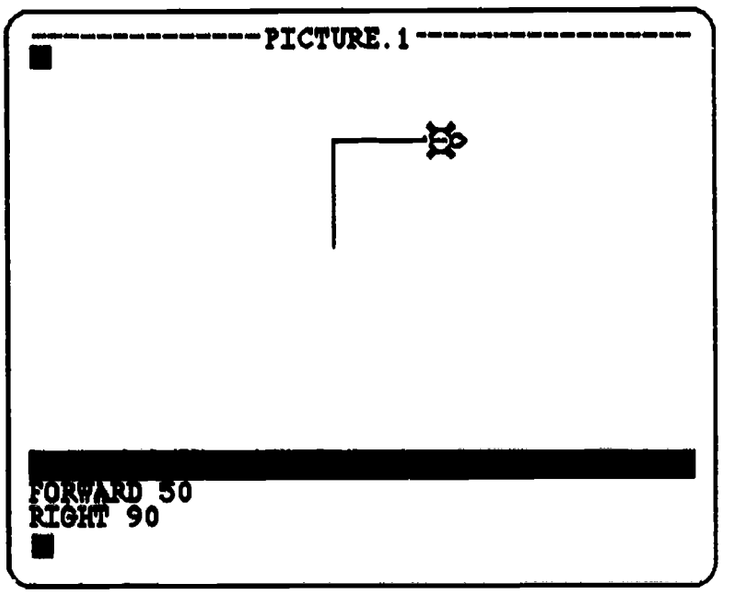
Next type

```
RIGHT 90 (Return/Enter)
```

or use the abbreviated form

```
RT 90 (Return/Enter)
```

Notice how the turtle turns 90 degrees to the right. Using the up arrow key, move the cursor that is at the bottom of the screen until it is on the "FORWARD" line. Press Return/Enter. Does your turtle drawing look like this?



Do you see that you don't have to retype an instruction to use it again? Can you finish drawing the square without typing any more commands?

Primitives and Instructions

You have now learned three LogoWriter *commands*: NAMEPAGE, FORWARD, and RIGHT. Words that LogoWriter "understands" are called *primitives*. A command is one kind of primitive. You will learn about a second type of primitive, reporters, later in the book.

The word *instruction* is used to describe a line of Logo code. A correctly written instruction always consists of a command along with the inputs (if any) to that command. Thus, while FORWARD is a command, FOWARD 50 is an instruction. It is important that you learn the meaning of these three terms: command, primitive, and instruction. They will be used frequently throughout this book.

Now try typing

```
CG (Return/Enter)
```

to clear the screen. CG stands for Clear Graphics.

What happens if you type

```
FORWARD
```

and press Return/Enter? You see

```
forward needs more inputs
```

Logo is telling you that you must put a number after FORWARD to tell the turtle how far forward to move. Logo usually gives you helpful messages when you make a mistake. Read them carefully!

Below is a list of some of the commands that the turtle understands:

FORWARD *number* FD *number*	BACK *number* BK *number*
RIGHT *number* RT *number*	LEFT *number* LT *number*
PU (Pen Up)	PD (Pen Down)
CG (Clear Graphics)	

Experiment with these commands to see how each one works. Remember to press Return/Enter after each instruction. Use your journal to record ideas on what your are learning and your feelings about learning by experimentation. After you have experimented for awhile, decide on a design or drawing to make. If you make mistakes you can use CG to clear the page and start again. Remember that the commands you used are still available at the bottom of your screen in the Command Center.

Saving and/or Printing a Page

Whenever you have something on the screen you want to keep, press the Esc key. Esc causes LogoWriter to make your page a part of your Scrapbook. That is, it puts the page on your Scrapbook disk, using the name you have given it. Recall that the term "page" in LogoWriter refers to the part of the screen above the solid line; the instructions you typed below that solid line are in the Command Center. The Command Center is not saved with the page. You will learn how to save groups of instructions later in this book (Chapter 10).

When you press Esc to save a page, LogoWriter automatically takes you to the Contents page. To continue working with the page you have saved, simply select your page from the Contents page and you will see that it looks exactly as it did when you left it. If you haven't already done so, try saving your page right now.

If your computer is attached to a printer, you can print a copy of a page. To print the page, first be sure your page has been saved. At that time you will see the Contents page on the screen. Select the page you want to print and then type

```
PRINTSCREEN (Return/Enter)
```

Note: Your disk containing the LogoWriter language must be configured for use with the printer you are using. It is assumed that your course instructor has provided you with a disk that is properly configured for the printer you are using.

Frequently Asked Questions

In this and the remaining chapters we include a section on Frequently Asked Questions. These are the types of questions that beginners frequently ask when studying the material being presented in the chapter.

1. When I type FORWARD and a number, the turtle moves up the screen. Why doesn't it move to the right when I type RIGHT and a number?

   Answer: Beginners sometimes find the differences between the commands FORWARD and BACK, and the commands RIGHT and LEFT, difficult to understand. FORWARD and BACK are movement commands. They cause the turtle to take "turtle steps." RIGHT and LEFT are turn commands. The turtle doesn't move at all. Instead it turns about its middle, like a ballet dancer spinning on her toe. The number you give is the number of degrees in the turn.

2. I know that if I draw a line I don't want in my picture I can always start over by using CG or by getting a new page. Sometimes I want just to correct the work I have done so far. Is there a way to erase an unwanted line?

   Answer: It is not unusual to want to erase lines you draw by accident. It is possible to do this using PE (Pen Erase). You type PE and go back over the line you don't want. Then you must type PD to continue to draw. Unless you erase a line immediately after drawing it, it can be very difficult to erase it completely because it is tricky to place the pen exactly on top of where you have previously drawn. Generally it is better to start over rather than to try to fix several small errors on a drawing.

3. I worked really hard on a picture, but it was gone when I next tried to use my Scrapbook disk. What happened?

   Answer: Most likely you didn't save your page before you turned off the computer. Recall that pressing Esc saves your page. When you are finished working with LogoWriter, always be sure that you see the Contents page on your screen before you turn off the computer or let someone else use it. Since you must press Esc to get to the Contents page, any work you have done is saved. (You can always erase unwanted work at a later time.) When you save a page using Esc, only the graphics and letters above the solid line are saved. Text in the Command Center is not saved when you press Esc.

4. What are the important points that I need to remember about using spaces in LogoWriter?

Answer: Spaces are very important in LogoWriter. They are used to separate the parts of an instruction line in Logo. Thus, you must have a space between a command like FORWARD and the number after it. You also need a space after NAMEPAGE and before the page name. On the other hand, there must not be a space after the quotation mark (") in the name of a page. If you get error messages that you don't understand, look carefully at the spacing in what you typed. The problem may simply be in your spacing.

5. How do I delete a page I no longer want?

   Answer: Go to the Contents page. Move the cursor to the page you want to remove. Use the Erase to End of Line key combination. (See the Appendix for the keys to use on your computer.) Caution: A page deleted from the Contents page cannot be recovered.

Bugs and Debugging: Learning to Make Mistakes Through Logo

Every computer programmer makes errors. Even the very best of professional computer programmers write programs that contain bugs. The program writing process is one of striving to write bug-free programs, learning to detect bugs, and learning to correct bugs. You cannot be a successful programmer unless you can function well in an environment full of bugs needing debugging.

Think about yourself and others you know. Do you know people who seem almost deathly afraid of making a mistake? For example, do you know students who won't give an answer out loud in class for fear of being wrong? Do you know students who have trouble writing because they don't like to have erased spots where they have corrected errors? (A word processor certainly helps such people!)

Very young children are not afraid of making mistakes. They learn through trial and error as a result of appropriate feedback. It is natural to make mistakes. Mistakes are an essential part of the learning process. The nature of children's learning experiences gradually shapes how they learn. Some young people eventually "learn" that it is better to not try than to face the consequences of failure. This can be a major handicap to learning. Writing and debugging Logo programs is one way to become more comfortable with making errors and correcting these errors.

One way to find the source of a bug is to "play turtle." Children learning Logo are taught to move like a turtle. They walk around the floor, making the turns and taking the turtle steps just as if they were a computer turtle. They learn to picture their bodies as being like the screen turtle. This is a powerful aid to debugging and to learning. Try it–it works for adults as well as for children!

As you work with Logo, you will find that there are two major categories of bugs you will encounter over and over again:

1. Errors the computer can detect. These are errors that cause the computer to produce an error message, such as "I don't know how to..." Often this results from an error in keyboarding, an error in the spacing in an instruction, an error in the punctuation or in the spelling of words in an instruction, or a misunderstanding of the details of how a specific command must be written. Such errors in the "grammar" of Logo are usually called *syntax errors*.

2. Errors the computer cannot detect. These errors do not result in an error message, but they produce an incorrect result. It is easy to keyboard FORWARD *50* when you actually meant FORWARD *60*. The computer has no way of knowing what you had in mind. It is easy to use LEFT 90 when what is really needed is RIGHT 90. Again, the computer has no way of

knowing what is needed. However, you can look at the results produced on the screen and detect your error. We will call these errors in meaning *logic errors*.

You may want to deliberately make some programming errors so you can gain increased skill in detecting and correcting them. For example, try FORWARD TWENTY or RIGHT30. You may want to write in your journal some personal feelings about what it is like to make a mistake. How do you detect "bugs" in your thinking and problem solving activities in school and outside of school. What could you do to get better at this process?

Computers and Problem Solving: Logo and Learning New Things

One of the key ideas in problem solving is domain specificity. This is a fancy way of saying that you must know a great deal about a specific problem area in order to solve problems in that area. Suppose you are going to use Logo to solve an art problem. You must know about two domains—Logo and art.

Think about domain specificity as you learn Logo. Initially you know very little about the Logo language and the Logo programming environment. You have no way of knowing that FORWARD refers to moving a "turtle" up the screen a certain number of turtle steps. You have no way of knowing that RIGHT refers to turning the turtle to the right a certain number of degrees. The words FORWARD and RIGHT already have meaning to you in other domains, and now you are learning their meaning in a new problem solving domain.

Much of your initial learning is a sequence of trial-and-error efforts. You are being asked to learn a large number of new vocabulary words and ideas in a very short period of time. You may frequently reread portions of this text. The smallest error in your memory in following instructions or in keyboarding, is apt to produce a wrong result. Sometimes Logo gives you an error message and leaves you to figure out what you did incorrectly. At other times the pattern you have produced on the screen is not what you had in mind. You provide your own error message ("That doesn't look right.") and then attempt to figure out what you did wrong. You may be tempted to ask your course instructor or your fellow students for help each time something doesn't work as you expected. Eventually, however, you will become more self-sufficient.

Activities

1. Practice using the commands in this chapter. You might try drawing:

   - a rectangle that is twice as wide as it is high
   - an equilateral triangle (all three sides are the same length)
   - your initials
   - a border around the page
   - a line of dashes
   - a Morse code message
   - a bolt of lightning

   Use your imagination and feel free to change your mind as you work. You will find that, when working with Logo, you often start out with one goal and end up working towards another. For example, perhaps you begin making a letter M. Your drawing doesn't look much like an M, but it does look like a bolt of lightning. So you change your goal and work towards making a bolt of lightning. Keeping a journal of your progress, including changes in focus, may be help you examine your own learning style when working with Logo.

2. Think back over your first session with Logo. Be aware of how many new things you were being asked to learn in a relatively short period of time. What aspects of the book and the teaching/learning situation helped you? What aspects could have been improved? Write your reactions in your journal.

3. Discuss the questions in activity 2 with several other people in your class. Draw on journal entries such as you made in activity 1. To what extent are your classmates' responses the same? To what extent do they differ? People have differing learning styles. The authors of this book, the instructor of your class, and you may have differing learning styles. When your learning style differs from that of the book or the instructor, a dissonance results. This dissonance may hinder your learning. However, it provides you with an excellent chance to learn about learning and to learn more about teaching. Write your reactions to the these ideas about learning style in your journal.

Chapter 4
Using REPEAT and Turtle Move Mode

You have learned how to make pictures with the turtle. You have also learned how to save pages on your Scrapbook disk. You have an initial level of confidence in your ability to function in a Logo learning environment. Hopefully you have had some fun with Logo. Now you can begin adding more Logo commands to your new vocabulary and increasing the scope of the types of problems you can solve using Logo.

Each new command that you learn gives you more power. It allows you to accomplish things that you could not previously accomplish or that were previously quite difficult to accomplish. The REPEAT command is particularly useful. It gives you the power to tell the computer to do something over and over again.

Using REPEAT

Get a new page, name it, and then type

```
FORWARD 50
RIGHT 90
```

Remember to press Return/Enter after each line. If you want to make a square, you can repeat each of these two commands three more times. One way to do this is to use the up arrow and the Return/Enter key. However, there is an easier way for you to have Logo repeat actions. Type

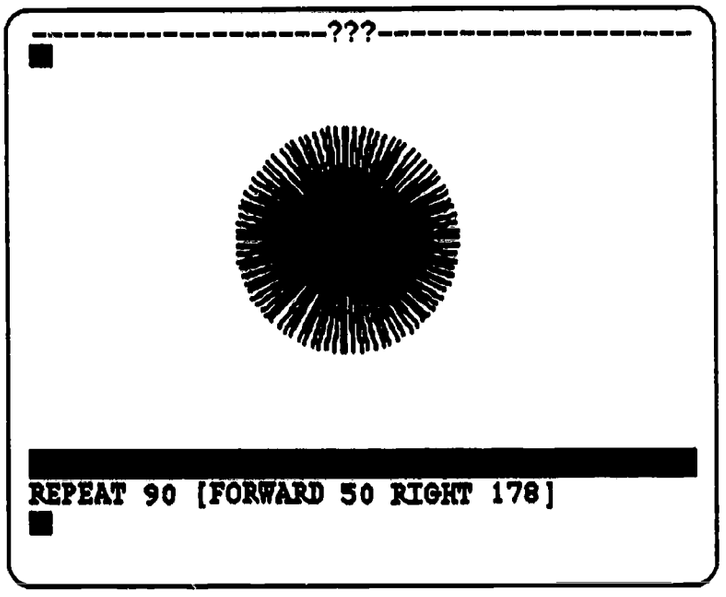
```
CG
REPEAT 4 [FORWARD 50 RIGHT 90]
```

What happens?

REPEAT is a new command. It must be followed by a number and then a list of instructions enclosed in square brackets. Try the following.

```
CG
REPEAT 90 [FORWARD 80 RIGHT 178]
```

Don't forget to press Return/Enter. You get a "fluffy" ball.



How about

```
CG
REPEAT 18 [FORWARD 20 RIGHT 45 BACK 10 LEFT 25]
```

or

```
CG
REPEAT 360 [FORWARD 1 RIGHT 1]
```

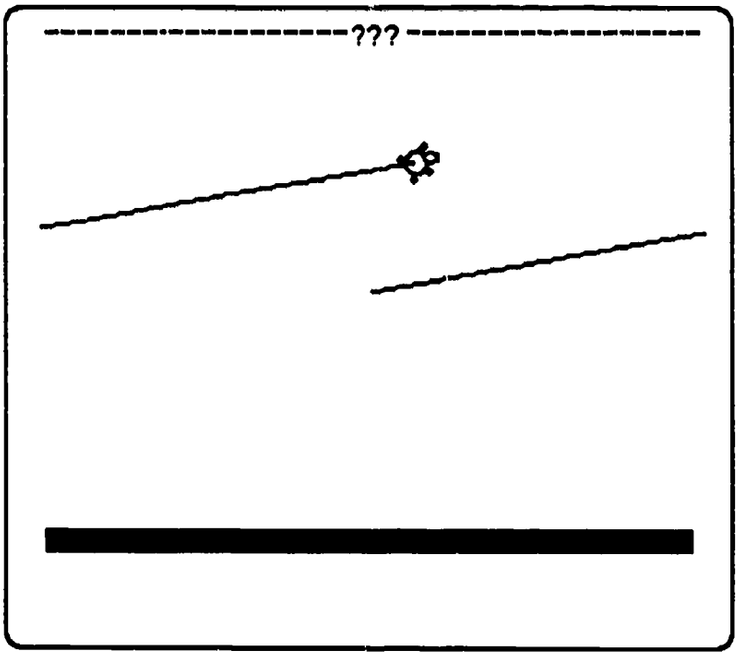
Take some time to try each of these examples as well as some ideas of your own.

Wrapping

Have you discovered what happens when the turtle goes past the edge of the page? If you type

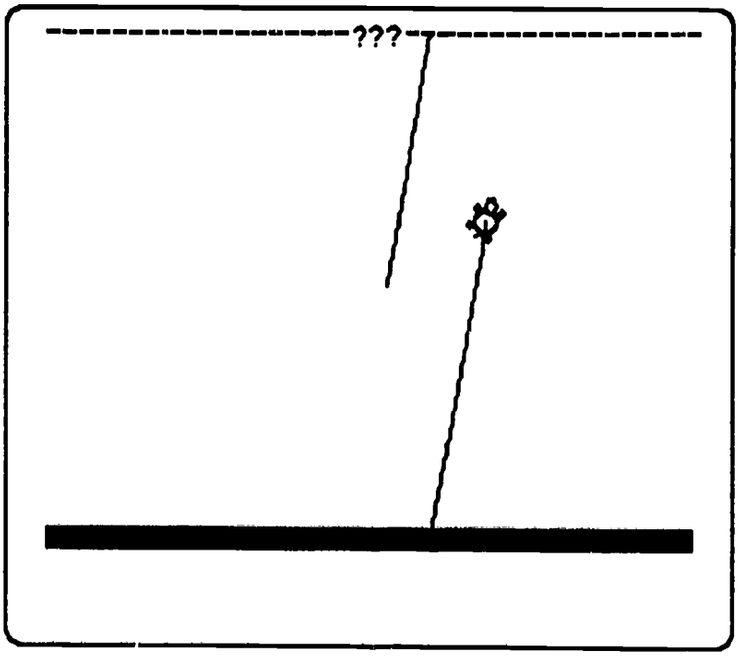
```
CG
RIGHT 80
FORWARD 300
```

you see that when the turtle goes off the page on the right, it reappears again on the left at the same distance from the bottom of the page. This is called *wrapping*.



If you type

```
CG
RIGHT 10
FORWARD 200
```



you see that the turtle wraps around the top and bottom of the page as well.

If a design you make would look better without the turtle showing, you can type

```
HT
```

for Hide Turtle. When you want to see the turtle again, you can type

```
ST
```

for Show Turtle.

Take some time to experiment with REPEAT and with wrapping. You can get some fascinating patterns and designs. Keep a record of your discoveries, as well as your thinking, as you work.

Renaming A Page

What if you save a design on your page by pressing Esc and then you change what is on the page and want to keep that *new* design as well? If you press Esc, the newest design will erase the previous one. *Esc always saves the current page using the name at the top of the page.* You can solve this problem by renaming the page.

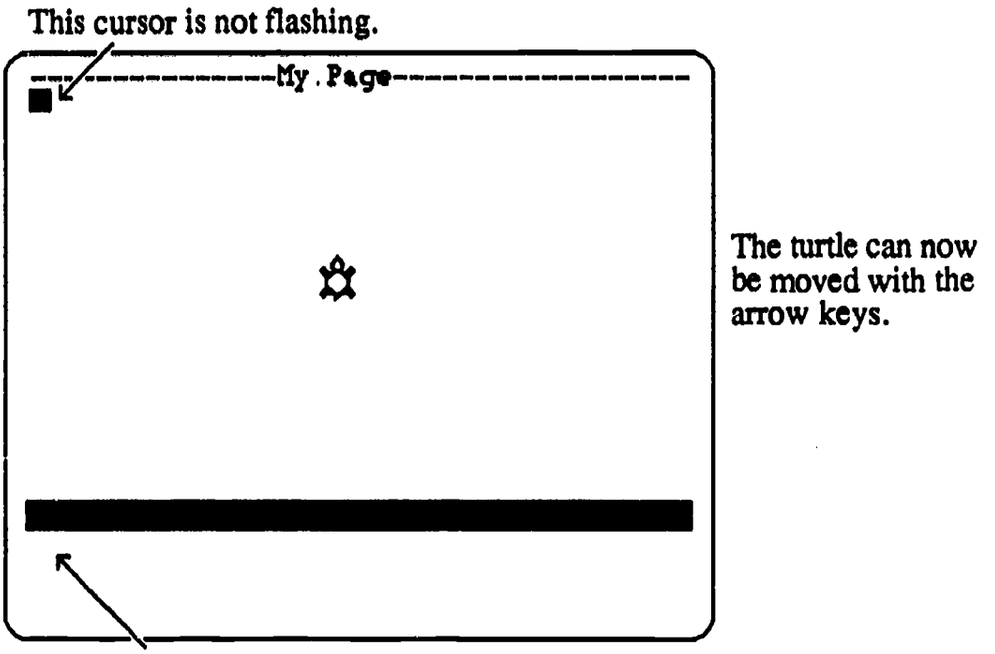
For example, suppose you create page PICTURE.1 and press Esc. Then, if you select PICTURE.1 from the Contents page again and change the design on the page to something you want to keep, you must change the name of the page in order to save both the old and the new design. Type

```
NAMEPAGE "PICTURE.2
```

You will see the new name at the top of the screen change. Next, press Esc to save the new page. When the Contents page appears, you see the names of both pages. You can then choose PICTURE.1, PICTURE.2, or a new page.

Turtle Move

You have been moving the turtle using the FORWARD, BACK, LEFT, and RIGHT commands. There is another way to move the turtle on the page. Hold down the Turtle Move keys. (See the Appendix for the specific keys to use with your computer.)



Now use the arrow keys to move the turtle away from the center of the page. *To leave Turtle Move, press Esc.* When the cursor is again flashing in the Command Center, type

```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

You now have a square that is not in the center of the page. You can use Turtle Move to place designs wherever you want on the page.

Frequently Asked Questions

1. What happens if I use the parentheses (and) or braces { and } instead of the square brackets [and] in a REPEAT command?

   Answer: This is the sort of question you should answer for yourself. Try it and see what happens. Parenthesis and square brackets have special meaning in Logo. Be careful to distinguish between them when you learn new Logo commands.

2. I lost my cursor. Where is it? Where am I working?

   Answer: It is easy to get confused as to which mode you are in. When you are in Turtle Move Mode, there is a cursor in the upper left hand corner of the page and it is not flashing. (You will learn about that cursor later.) In some versions of LogoWriter, the cursor in the Command Center is missing; in others it is there but not flashing. To get back to the flashing cursor in the Command Center, press Esc. Esc exits Turtle Move Mode.

   Don't worry that you might press Esc too often. If you have named your page, LogoWriter will simply save your work and put you on the Contents page. Select the page you were working on and continue your work. If you have not named your page, LogoWriter will ask you to name it. If you wish, you can ignore this message and continue your work.

Bugs and Debugging: Learning Logo Becomes More Complex

In the initial stages of learning a programming language like Logo, it is relatively easy to detect and correct bugs. This is because the initial learning focuses on learning the syntax and meaning of a number of commands and how to use them in Logo instruction lines. In essence, you are learning some vocabulary and how to write the very simplest of sentences using the vocabulary. It is easy to correct the types of errors you make when using one word or very short sentences written with very few words.

However, LogoWriter contains over 200 primitives (vocabulary words). Moreover, as you proceed in learning Logo, you will find that the primitives can be combined into more and more complex instructions. Furthermore, sequences of instructions can be grouped together in order to solve increasingly complex problems.

Thus, you will see a gradual shift in the types of bugs that you create. As you learn more primitives, you will sometimes forget some of the relevant details. For example, instead of using CG for Clear Graphics you might use CS, thinking that what you want to do is Clear Screen. But there is no CS primitive in LogoWriter. A good memory for such naming details is helpful but is certainly not necessary. You can aid your memory by keeping a written list of the primitives you have learned. You might make a copy of the Quick Reference in the Appendix of this book, and then highlight new primitives as you learn them.

Later on in your learning, you will begin to write longer instruction lines to attack more complex problems. You will make more and more logical errors that don't produce the results you want. In early stages of learning Logo, almost every instruction you write produces a result you can see on the screen. If you continually check the screen results against what you expected to produce, you will be able to detect these types of bugs immediately.

Later in this book you will learn how to write long sequences of instructions that the computer carries out only after you have completed writing the entire sequence. At that time you will be faced with the task of finding bugs that may be due to a variety of errors of different types. Furthermore, the errors may interact with each other to produce some very strange results. Determining the source of such errors can be very challenging.

Computers and Problem Solving: REPEAT and Human Thinking

A computer is a very fast machine. Even an inexpensive microcomputer can multiply two big numbers together in less than a thousandth of a second. The very fastest of modern computers can carry out more than a billion arithmetic operations in a second.

Suppose that the idea of repetition did not exist in computer programming. Then a computer program would be a linear sequence of instructions. It would take a very long program to keep a computer busy for even a few seconds. It is easy to see why the idea of repetition is very important in computer programming. Most computer programs involve having a computer do a great deal of repetition, perhaps with small modifications between repetitions.

This also points out a major difference between humans and computers. The human mind is not good at doing the same task over and over again quickly and without error. The mind soon becomes bored! However, a computer can do the same task over and over again hour after hour. It can do this without error, and without getting bored.

This ability of computers brings a new dimension to problem solving. This is a challenge to teachers and to students. Where in your schooling did you learn to think about the possibility of

repeating a certain action many thousands of times in order to solve a problem? Are there certain types of problems that can be solved that way? Is the basic nature of solving problems by trial and error changed by computers?

Every computer programming language contains provisions for telling the computer to do something over and over again. Different languages contain different provisions for this. The REPEAT command in Logo is but one of several ways to cause repetition to occur in Logo.

Activities

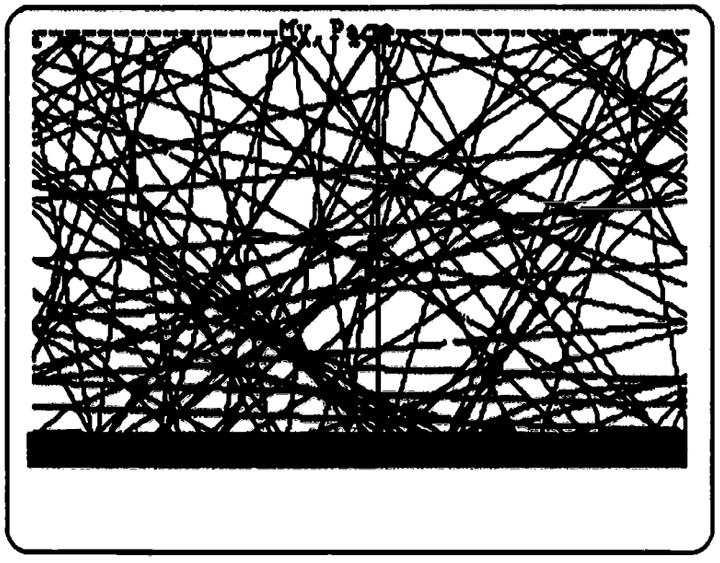
1. Create a drawing that is not in the center of the page. When you are satisfied with your drawing, press Esc to save it and then type PRINTSCREEN to print it using the printer. If you can't think of anything else, use REPEAT to make a fluffy ball in the upper left corner of the page and a star in the lower right corner of the page.

2. Using REPEAT, create a flower blossom. Use Turtle Move to create a garden of flowers all over the page.

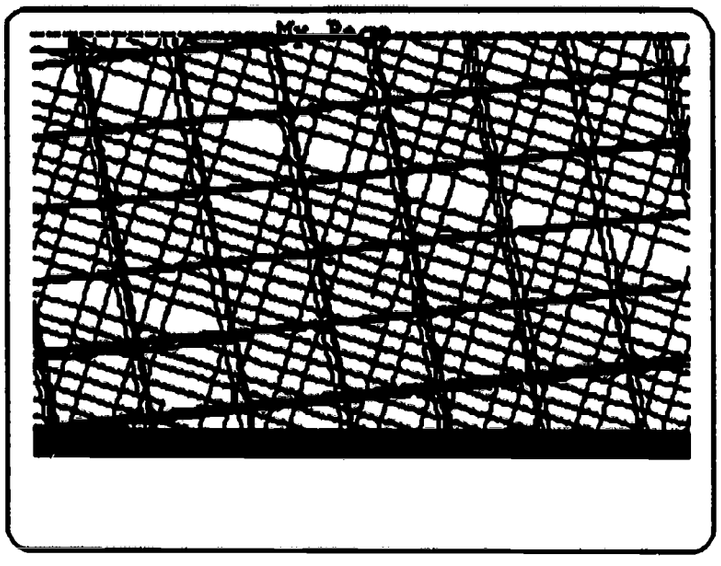
3. Students often become fascinated with the wrapping feature of Logo and spend many hours making "plaids." Experiment with instructions like

```
REPEAT 20 [FORWARD 1000 RIGHT 17]
```

What kinds of instructions produce patterns that look like this first picture?



Can you produce designs that look like this second picture with a single REPEAT instruction?



What is so fascinating about these designs? Can you find interesting patterns? Can you think of curriculum-related activities that would use this wrapping phenomenon? Write your thoughts in your journal.

4. Think about the idea of repetition in education. When you were in school, did you learn to think about repeating a certain action many thousands of times in order to solve a problem? How many times did you practice this approach to problem solving while you were in school? Think of examples of problems that might be solvable by extensive repetition or extensive trial and error. Does the computer change the way we should teach problem solving? Spend some time writing in your journal about these ideas.

Chapter 5
Color and Random

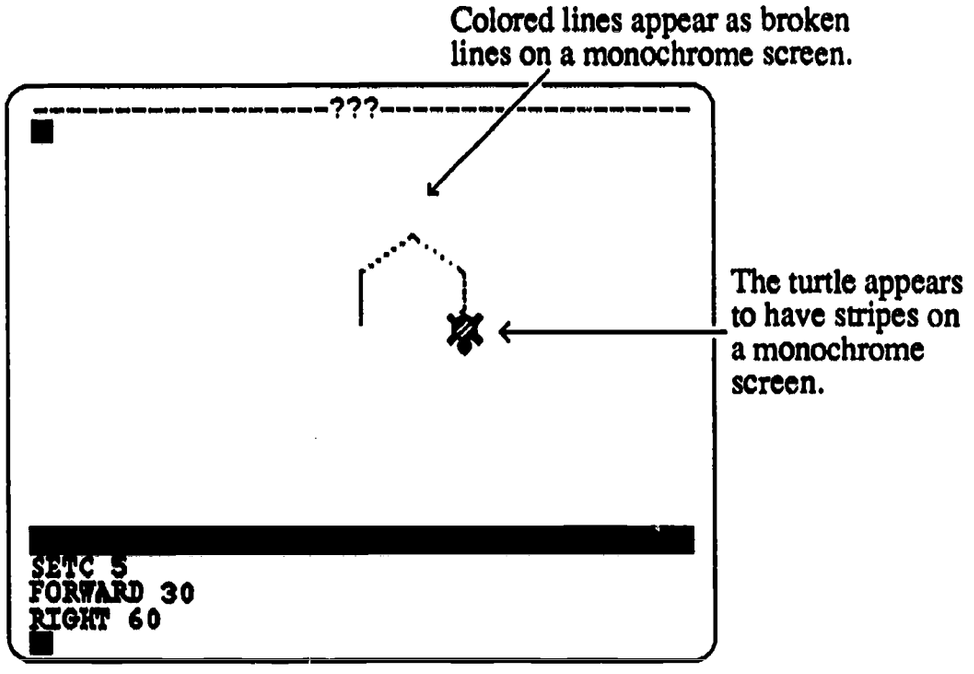
Drawing in Color

So far you have created drawings with the turtle that did not make use of color. The turtle can also draw in colors if you have a color monitor. Try this series of instructions and watch carefully what happens. (You might also want to jot down the six numbers you use and the colors they represent.)

```
SETC 2
FORWARD 30
RIGHT 60
SETC 3        <—  Use arrow keys to move up and the Delete key to change the number from above.
FORWARD 30        Use the arrow keys so you don't have to retype the FORWARD and RIGHT commands.
RIGHT 60
SETC 4
FORWARD 30
RIGHT 60
SETC 5
FORWARD 30
RIGHT 60
SETC 0
FORWARD 30
RIGHT 60
SETC 1
FORWARD 30
RIGHT 60
```

Did you notice that the turtle changes to the color of the pen it is carrying? What is the color number for white? What is the color number for black? What happens if the turtle turns black? What other color numbers did you discover? How many colors are available on the computer you are using? (You may need to do some experimentation to answer this question. The number of available colors varies with the type of computer system being used.) Keep in mind that the color of the turtle represents the color of the *pen* it is carrying and thus the color of the line the turtle will draw.

NOTE: If you are using a monochrome screen and change the color of the turtle, the turtle will appear to have stripes and the lines that are drawn will be broken.



Changing the Background Color

Not only can you change the color of the pen using SETC, you can also change the color of the background. For example,

```
SETBG 3
```

changes the color of the entire page. Experiment with different background colors.

On some computers, you will discover that when you draw on a colored background with a colored pen, you occasionally get some odd effects. The colors aren't what you expect them to be. That is not something you are doing wrong. It has to do with how color is represented inside your computer. For best results you will want to use black or white pens on colored backgrounds and use colored pens only on black or white backgrounds.

Using RANDOM

Everything you have told the turtle to do so far has been exact and totally predictable: "go forward 50 steps," "turn left 57 degrees," or "draw in pen color 3." It is also possible to have the turtle behave in ways that you cannot totally predict. You can use Logo to create random numbers and you can use instructions that contain random numbers. Random numbers are numbers that have no apparent pattern to them. Think about rolling a die. This produces a random outcome between 1 and 6. Think about flipping a coin to produce a random number which is a 0 or a 1, with heads standing for a 0 and tails standing for a 1.

The Logo primitive RANDOM is used to create random numbers. We can use another new primitive, PRINT, to see the results of using RANDOM. Try, for example

```
PRINT RANDOM 10
```

You see a number on the page. Use the arrow keys to repeat this instruction or type

```
REPEAT 20 [PRINT RANDOM 10]
```

Twenty random numbers appear on the page. (You can type CT for Clear Text to remove numbers you don't want.) Experiment with this instruction. Do you see that when you use the number 10 as input to RANDOM, you get numbers from 0 to 9?

RANDOM and PRINT are both primitives. They are, however, two different types of primitives. PRINT is a command, like FORWARD and RIGHT. It causes an immediate effect–putting text on the page. You must provide an input to PRINT or you will get an error message. On the other hand, RANDOM is a *reporter*. (We will say more about this later in the chapter.) It produces a number and reports it in a form that can serve as an input to a command.

Next try

```
CG
FORWARD RANDOM 50
```

Repeat these two instructions a number of times. Do you see that the turtle moves different amounts at different times? RANDOM reports to FORWARD some number from 0 to 49 and then the turtle moves forward that number of turtle steps. That is, if RANDOM 50 reports the number 37, then the turtle follows the instruction FORWARD 37. If RANDOM 50 reports 11, then the turtle moves forward 11 turtle steps.

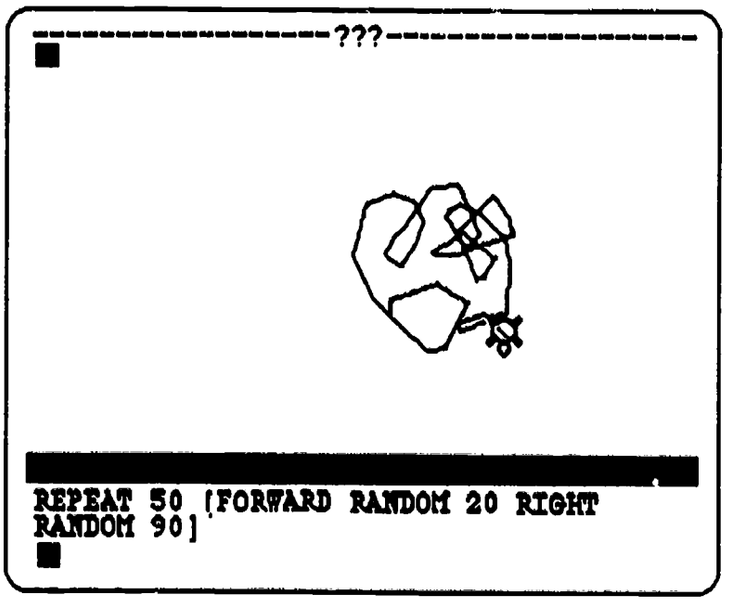
What happens if you try to make a square, but use random length sides? Try it!

```
CG
REPEAT 4 [FORWARD RANDOM 50 RIGHT 90]
```

You can send the turtle on a random walk by typing something like

```
REPEAT 50 [FORWARD RANDOM 20 RIGHT RANDOM 90]
```

Did your design look anything like this?



RANDOM is different from the Logo primitives you have seen so far. If you type

```
RANDOM 50
```

LogoWriter responds with

```
I don't know what to do with some.number
```

This message appears because RANDOM is a reporter. It produces a value and reports it back to Logo.

To use the RANDOM reporter, you must always tell Logo what to do with the number it produces. Our previous examples show that the computer can report the value to a movement command, such as FORWARD, or a turn command, such as RIGHT. You also saw that you could use a PRINT statement to see the results produced by RANDOM.

Take time to experiment with RANDOM. Can you get some other interesting effects using RANDOM? How about picking random colors? That certainly is possible. Recall that the colors are numbered beginning with zero. So, if you use RANDOM 6, it will report a number between 0 and 5, which is exactly what you want if you are using a computer with six colors. You can change the background color each time the turtle draws a line. Try this:

```
CG
SETC 1
REPEAT 4 [SETBG RANDOM 6 FORWARD 50 RIGHT 90]
```

What about random pen colors? You can get some beautiful patterns by simply changing the pen color randomly each time you move the turtle. For example,

```
CG
REPEAT 90 [SETC RANDOM 6 FORWARD 80 RIGHT 178]
```

You can get flashy effects by typing:

```
REPEAT 10 [SETBG RANDOM 6 WAIT 5]
```

Note that we have introduced a new command called WAIT without an explanation. Can you figure out what it does? Try some experiments to see if you can determine what the number after WAIT represents. Think about the problem solving strategies that you might use to answer this question. Perhaps you'll want to record your observations in your journal.

When you are working with color, the easiest way to reset the screen is to use

```
RG
```

The Reset Graphics command puts the turtle in the center of the page, sets the background to black, and the pen color to white.

Frequently Asked Questions

1. How can I tell if I have a color monitor?

   Answer: If you try several different numbers after SETBG and don't get different colors, then you are using a monochrome screen or the color is turned off on your color display monitor. There is no way to get color on a monochrome monitor.

2. Why does RANDOM 6 report one of the integers 0, 1,..., 5 rather than an integer in the range of 1 to 6?

   Answer: This is an "arbitrary" decision that was made by the creators of LogoWriter. In many cases, mathematicians seem to like to start counting at 0 rather than at 1. This has carried over into a number of different programming languages. Note that 1 + RANDOM 6 will be a random integer in the range of 1 to 6.

3. What is the time unit used with the WAIT primitive?

   Answer: It is 1/20 of a second. WAIT 10 specifies a wait of 10/20, or 1/2 second. WAIT 60 specifies a wait of three seconds.

4. Why do I get the same sequence of random numbers if I use the statement REPEAT 20 [PRINT RANDOM 10] immediately after turning on the computer?

   In some versions of LogoWriter, when the computer is turned on using Logo, the sequence of random numbers is always the same. There are a number of ways to solve this problem that you will learn if you continue your work with Logo.

Bugs and Debugging: Random Bugs Are Harder to Find

The idea of a reporter adds a new level of complexity to your Logo learning. A reporter, all by itself, does not produce a result that shows as part of a drawing you are making. Rather, it produces a result that is then used in conjunction with another primitive to make an instruction that can contribute to what appears on the page.

The reporter RANDOM also adds another debugging difficulty. Up to this point you have been able to have an exact picture in your "mind's eye" of what you expect a drawing to look like. You were able to detect errors easily because the screen display did not look like what you had in mind. It is much harder to get a good picture in your mind's eye of the types of results that will be produced when you make use of RANDOM. Thus, you might well have made an error in programming logic and be unaware of it.

There is no easy solution to this new error detection problem. You will want to think very carefully about what you want to produce with the instructions you are writing. You will want to write these instructions (often using pencil and paper) and think carefully about them. Are they really what you want? Will they produce the results you have in mind? You will want to think about ways of testing whether the results you have produced are correct. These are all important aspects of computer programming.

Computers and Problem Solving: Modeling Using RANDOM

One of the most important ideas in using a computer to help solve problems is figuring out ways to represent a problem on a computer. Logo is particularly powerful as an aid to representing graphically oriented problems. You can draw a picture on the screen, look at it to see if it is what you had in mind, and then easily change the picture.

When we represent a problem on a computer, we say we are developing a *computer model* for the problem. A computer model is different from a scale model or a paper-and-pencil model.

The primitive RANDOM is quite useful in developing computer modeling for certain types of problems. Suppose you are working with a class of 24 students and you want to select a student at random in order to ask that student a question. You can write each student's name on a slip of paper, put the slips of paper in a box, shake up the box, and draw out one slip of paper.

Here is a way to create a computer model of the slips-of-paper process. Number the students in your class with the numbers 0, 1, 2, ..., 23. When you want to select a random student, use

```
PRINT RANDOM 24
```

Now you have constructed a computer model of the process of drawing numbers out of a hat to select a student at random.

Maybe you think it is unnatural to number your students starting with 0. Instead, you want to number your students 1, 2, 3, ... , 24. Then use

```
PRINT 1 + RANDOM 24
```

Notice that RANDOM 24 will produce a random integer in the range of 0 to 23. Adding 1 to the result produces a random integer in the range of 1 to 24.

We have just developed two somewhat similar computer models for the process of selecting a student at random. Perhaps one seems more natural or better to you. There are many different models that can be developed for a problem. Generally, each has certain advantages and certain disadvantages. The study of computer modeling is a very important part of learning to use a computer to solve problems.

Think about the problem of designing a house and having it built. Here is a short list of some of the types of models that might be useful in this overall process:

1. Blueprints, to be used by the builder.

2. A drawing of the house and landscape, to be used by the landscaper.

3. A wiring diagram, to be used by the electrician.

4. A plumbing diagram.

5. A list of the amount and type of insulation and the nature of the building materials, to be used in an energy efficiency audit.

6. A financial analysis of your savings, income, and monthly payments you feel you can afford, to be used in obtaining a bank loan.

The list can easily be extended. Each type of model is useful in helping to solve some particular aspect of the problem of designing and building a house. Notice that it is possible to develop a computer model for each item on the list.

Computer modeling is very important for two reasons. First, computer models are often much easier to change than scale models or paper-and-pencil models. Second, a computer can often be used to help do part of the work involved in using a model. For example, suppose that we have a computerized energy audit for our house, and we decide to use bricks in place of wooden siding. With a computer model, we might produce a new energy audit in a few seconds.

Architects now make routine use of computer models. Programs have been developed that help an architect to design a building and then view pictures of it from different directions. This type of computer assisted design software is a powerful aid to solving architectural problems. As you learn more Logo you may want to try a simpler problem. Develop a computer model of a room arrangement, showing the windows, doorways, and furniture. Develop your computer model so that it is easy to make changes.

You might start thinking about this problem right now. Do you know enough Logo to effectively solve the problem? What are you lacking? Keep this problem in mind as you learn more Logo primitives. Eventually you will know enough Logo to develop a useful computer model for the room arrangement problem.

Activities

1. Create a page using color and RANDOM. You might

   - Send the turtle on a random walk with a random pen color at each move.
   - Draw geometric figures with flashing random background colors.
   - Fill the screen with randomly colored stars.

2. Experiment with repeatedly using each of the following:

   ```
   PRINT 1 + RANDOM 6 + 1 + RANDOM 6

   PRINT 1 + RANDOM 12
   ```

   Explain the similarities and differences between these two instructions in terms of the type of output you expect them to produce. Think of a non-computer model for each of them and discuss your thinking in your journal.

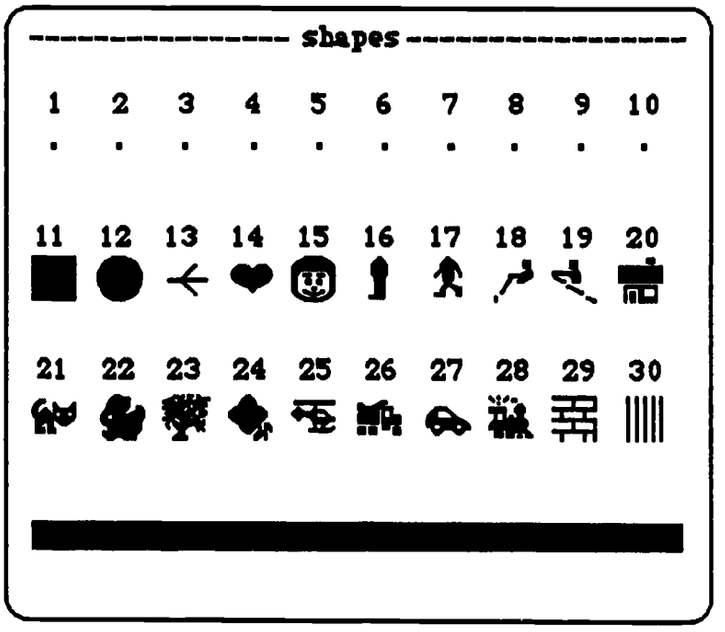
3. Design a simple model using Logo. Think about the advantages and disadvantages of using Logo versus other methods to create your model. Use your journal to jot down your thoughts.

4. Think of courses that you have taken where it would have been appropriate for the teacher to talk about different types of modeling. Was computer modeling mentioned? Write about this in your journal.

Chapter 6
Shapes and STAMP

Changing the Shape of the Turtle

Have you noticed the page Shapes in the list of pages on the Contents page? Maybe you even looked at it! If not, select Shapes and you will see a screen filled with small pictures. If you are using the Intermediate version of LogoWriter, your screen looks like this:



Numbers 1 - 10 each contain only a dot. Numbers 11 - 30 are a variety of shapes. (If you are using the Primary version of LogoWriter, numbers 1 - 25 are a variety of shapes while 26 - 30 are each a dot. See the Appendix for pictures of all of the shapes.) Is there one shape that you particularly like? Remember its number. Press Esc to leave the Shapes page and then get a new page.

Once you see the turtle on the screen, type

```
SETSH number.you.picked
```

Surprise! The turtle is no longer a turtle. Try another number. And another. The turtle can be "costumed" in any of the shapes shown on the Shapes page. Note that SETSH stands for "set shape."

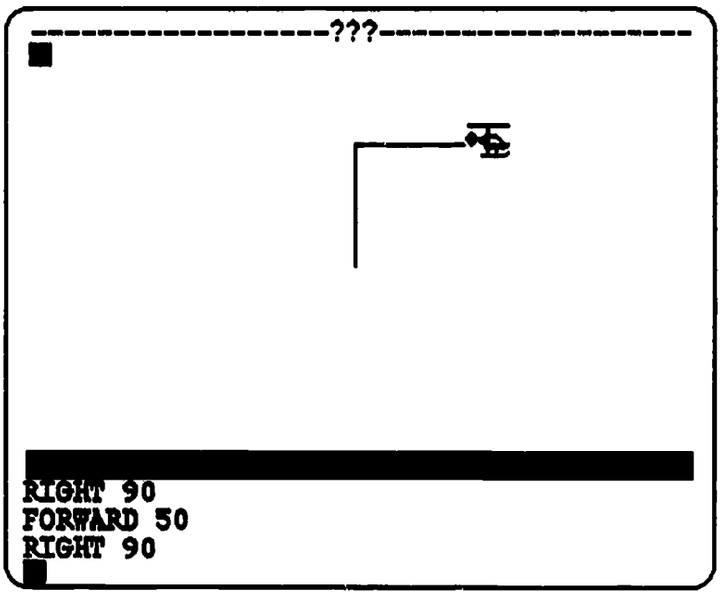
If you want the turtle shape back, type

```
SETSH 0
```

to return it to the original turtle shape.

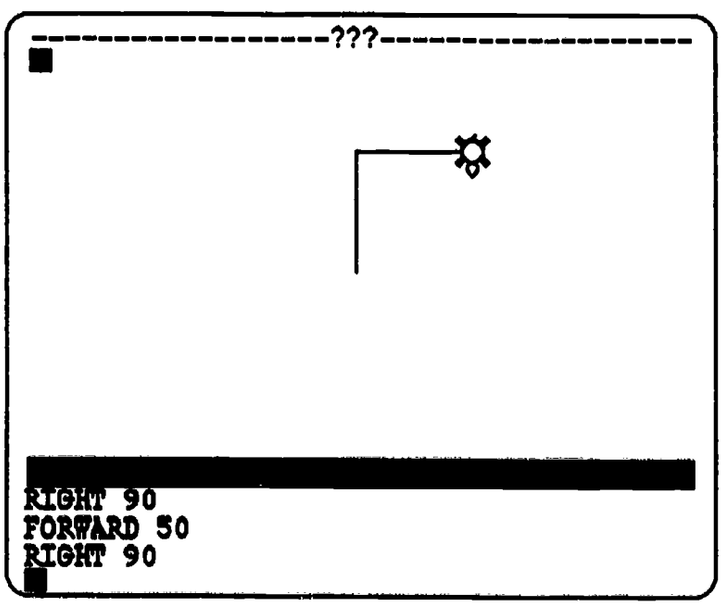
Next, with the turtle wearing a costume other than its familiar turtle shape, type

```
CG
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
```



Do you notice anything unexpected? If not, type

```
CG
SETSH 0
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
```



and compare the movement of the turtle shape and the other shapes. Do you see that the turtle shape turns to face in the direction that it is moving, but the other shapes do not? That can be a problem when you are trying to draw with the turtle and forget which way it is facing. Of course, you can always use

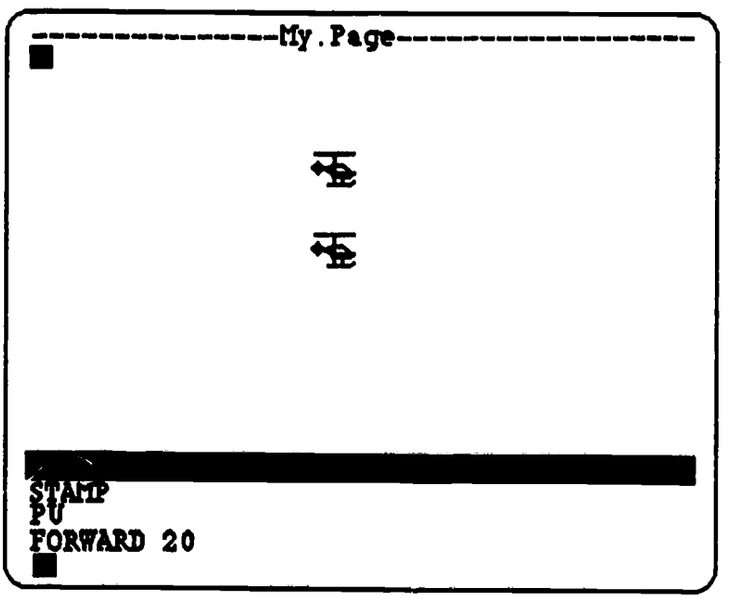
```
SETSH 0
```

to see where the turtle is facing. Generally, it is a good idea to leave the turtle in its normal "turtle costume" until you get a shape placed where you want it on the page.

Stamping the Turtle Shape

Not only can the turtle draw, but it can behave like the rubber stamps that people use to stamp dates or mark packages. The turtle can stamp only its current shape. Try this.

```
CG
SETSH number.of.your.choice
PD
STAMP
PU
FORWARD 20
```



You should now have two "copies" of the turtle shape. Which one is the actual turtle that is carrying the pen? Type

```
FORWARD 20
```

and you will see that one of the turtles moves and the other stays put. Another technique for discovering which turtle is active and which is a stamped copy is to use HT. The active turtle will disappear and the stamp will not change. You can then see the active turtle again by typing ST. As you can see, the stamp of the turtle shape will remain on the screen until you type CG.

Take note of the sequence of commands to get a stamped copy of the turtle:

- Put the pen down.
- STAMP the shape.
- Lift the pen (so there is no trail).
- Move the turtle (so you can see the stamped copy).

You can stamp any number of shapes in any number of places on the screen. Try it!

Frequently Asked Questions

1. Why does the turtle shape turn when I use a turn primitive, but the other shapes do not turn?

   Answer: LogoWriter has stored in the memory of the computer many pictures of the turtle facing in different directions. As you rotate the turtle-shaped turtle, LogoWriter uses the appropriate picture. To store many different versions of each LogoWriter shape would take much more memory than is available on the smaller computers used to run LogoWriter.

2. How can I tell which way the turtle is facing when I am using a turtle shape to draw?

   You can always find out which way the turtle is facing by typing

   ```
   SETSH 0
   ```

   In fact, when you are drawing with the turtle, it it a good idea to always use the regular turtle shape and then change to the costume you want to appear on the page just at the time you are ready to STAMP it.

3. STAMP is broken. What am I doing wrong?

   Answer: If STAMP does not seem to work, you probably forgot to put the pen down. Get into the habit of using the sequence

   ```
   PD
   STAMP
   PU
   ```

   so that STAMP works correctly and you don't accidentally draw lines after you stamp a shape.

   Occasionally STAMP won't work because of the turtle's position on the screen. This occurs because of the way the screen image is stored in the memory of some kinds of computers. Should this happen to you, simply move the turtle a few turtle steps and try again.

4. My turtle disappeared? Where is it?

   In some versions of LogoWriter, if you stamp a turtle on top of an already stamped turtle shape or move the turtle on top of a stamped turtle shape, both may disappear. Don't panic. Just move the turtle.

   If moving the turtle doesn't work, try typing

   ```
   ST
   ```

If the turtle is still missing, use SETC to change the color of turtle so that it is different from the color of the background.

5. How can I erase a shape I have stamped without erasing the entire screen?

Answer: You can erase a stamped turtle shape by setting the pen color to the background color and stamping on top of the shape you don't want. It can be difficult to get a shape in exactly the right place, however. Sometimes it is easier to use the square shape (number 11 in the Intermediate version of LogoWriter). You might type

```
SETSH 11
```

then use Turtle Move to place the turtle, set the pen color to the same color as the background, and then stamp the shape.

```
SETC BG
PD
STAMP
PU
```

These steps can be confusing because you can't see the turtle after you type SETC BG. When you have finished typing PU, be sure to set the pen color (SETC) to some other color, such as white. Incidentally, this same technique can be used to remove lines you no longer want in your drawing.

Bugs and Debugging: Turtle Shapes and Mental Models

In the previous chapter you encountered the reporter RANDOM that contributes to debugging problems. Part of the difficulty is that there may be a difference between what you have in your mind's eye and what the computer display actually looks like. This chapter adds a somewhat similar difficulty.

In this chapter you have learned about shapes and how to give the turtle a different costume. Unfortunately, you do not see the turtle with a new costume rotate on the screen when you make use of the RIGHT and LEFT commands. That is, the direction the turtle is facing (called the heading) changes, but the actual display of the costumed turtle does not change. In your mind you think of the costumed turtle as having a particular heading. However, on the screen the actual costumed turtle is not likely to appear to have that heading.

Clearly this contributes to confusion. One way to deal with this was suggested earlier in the chapter. Do all of your drawing using the standard turtle shape, until you are ready to stamp a costumed turtle. Change to the desired shape, stamp the costumed turtle, and then change back to the standard turtle.

This illustrates a very important idea in computer programming. You, your textbook authors, and your teacher can anticipate in advance some of the types of bugs you may produce. You can learn programming techniques that avoid the bugs, or make them less likely. In your journal you may want to keep track of the bugs you make most often. Gradually you will develop techniques for avoiding these bugs.

Computers and Problem Solving: Building on the Work of Others

Take another look at some of the shapes you have stamped. These are carefully drawn figures. You can use these shapes even if you don't have the artistic talent or the time to draw them

yourself. It doesn't take very long to learn how to use Logo shapes, and they can be very useful in producing pictures that are pleasing to you.

This illustrates a very important idea in problem solving—learn to build on the previous work that other people have done. This is one of the reasons that computers are so important. Suppose there is some problem that comes up quite often and that a computer can solve or help solve. Then someone can write a program to solve or help solve the problem. Other people can use the program. They can build on the work of the original programmer even if they don't know how to write computer programs themselves.

You might ask how this differs from just writing a book about how to solve the problem. In some ways writing a book on how to solve a problem is quite a bit like writing a computer program to solve a problem. In both cases the goal is to preserve and make available the knowledge about how to solve a particular type of problem. In both cases the goal is to help other people solve the problem. The difference is in the amount of learning and effort needed by the user. It takes a lot of time and effort to read a book and to learn what it says well enough so that you can do it yourself. If the same information is in a computer program, it may take very little time and effort to learn to use the computer program. Instead of learning to do all of the work to solve the problem by hand, you merely learn to tell the computer to solve the problem. The computer solves the problem by following the instructions in the computer program.

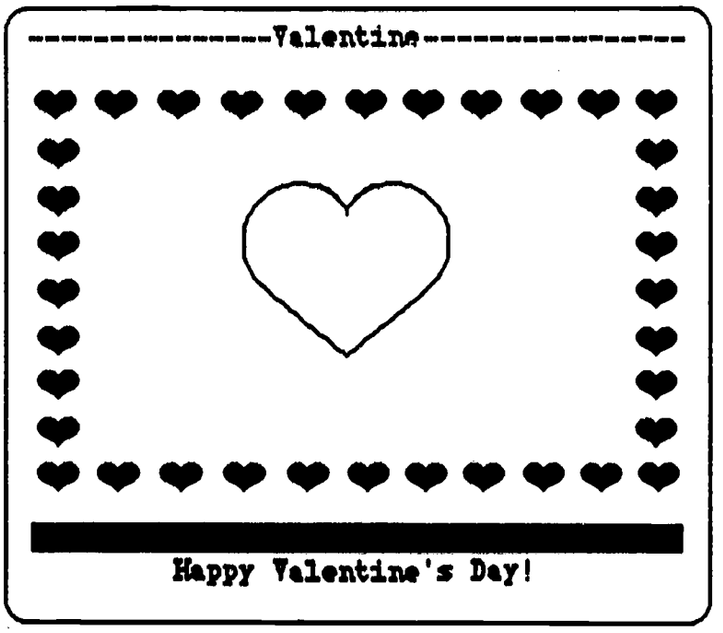
This is a simple idea, but it is the foundation for a revolution in education. If a computer can solve a type of problem that we currently teach students to solve by hand, what should we be teaching students about solving this type of problem? This is a very difficult question with no simple answer. However, there have already been major changes in some curriculum areas based on computers. Students are taught methods for solving certain problems "by computer," and are no longer taught methods for solving the same problems "by hand."

The areas of graphics design and mechanical drawing provide excellent examples. The ability to use different shapes for the turtle in Logo gives a hint of some of the computer's power to aid in doing graphic art work. Such computer aids are now routinely used in the world of business and industry. Many high school and college courses used to teach mechanical drawing and graphics design using only pencil and paper have been drastically changed because of computers.

Activities

1. By combining designs made with the turtle and stamped shapes, you can create a variety of interesting pages. For example, you might create a

   - Postcard with a picture.
   - Valentine's card bordered with hearts.



- Cover for a report.
- Design for a campaign poster.
- Decoration to go on the locker of a friend.
- An overhead to be used for class or for a report.

2. Try combining what you know about REPEAT with what you know about stamping shapes to create a page with a border of shapes around the edge. How many REPEAT statements did you need to use? How did you decide on the spacing of the shapes? Keep a record in your journal as you work. What problem solving strategies did you employ?

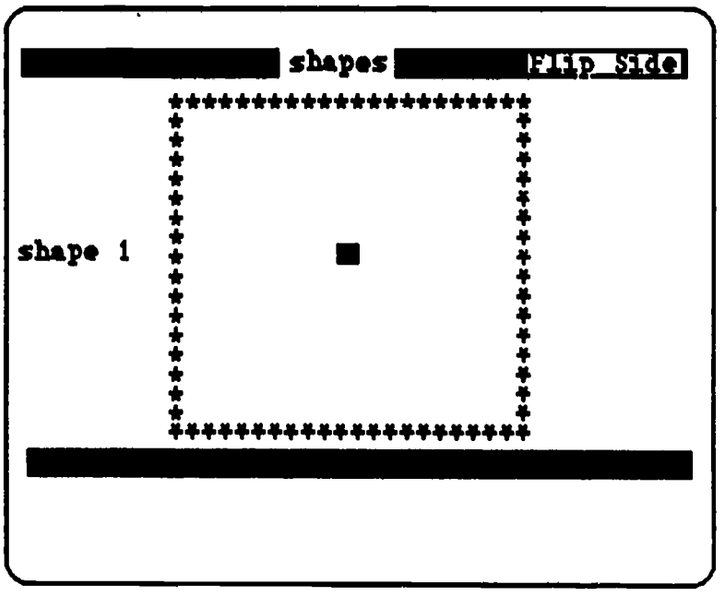
3. Have you been keeping track of the bugs you have encountered in your work with Logo? Take some time to look back through your journal. Do you see patterns in the kinds of problems you have? Are you finding your work with Logo to be getting easier? Harder? About the same? How do you think you can learn to manage your own debugging process better?

4. Think about the "by hand" methods we now teach students to use in schools. How do you think the computer might change what we now teach? Do you think theses changes are positive or negative? Do you have an area of "by hand" expertise where computers are now routinely being used? If so, what are your feelings about the time you spent in learning to do it by hand? Write your thoughts in your journal.

Chapter 7
Defining New Shapes

When you were working with the Shapes page in the last chapter, did you wonder why part of the shapes were just dots? In this chapter you will get the answer.

Go to the Shapes page and use the Flip keys to flip to the back of the Shapes page. (See the Appendix for the keys used on your computer.) The "back" of the page shows you an enlargement of shape 1. For the Intermediate version of LogoWriter it will look something like this:



Use the Next Screen and Previous Screen keys to move from shape to shape (see the Appendix). Take a look at all the shapes.

Go to a shape that contains just a dot. Use the arrow keys (Up, Down, Right, and Left) to move around within the shape. How many "blocks" across is the shape? How many up and down?

Next, press the space bar. If you are on a "colored in" block, it disappears. If you are not on a block, a block appears. See if you can make one of your initials in the shape grid.

When you have finished your initial, flip the page back. Do you see that your drawing appears on the front of the Shapes page now? Remember the number of the new shape you have created. Press Esc to leave the Shapes page, get a new page, and type

```
SETSH number.of.shape.you.created
```

The turtle is now the shape you created! Your new shape has become part of the Shapes page. Name your page, then type

```
SHAPES
```

to go back to the Shapes page. Pick one of the predefined shapes and change it. Perhaps you would like to change the way the kitten looks or make the car a bit more sporty. When you are finished, flip the page to see your revised shape. When you press Esc, LogoWriter returns you to the page you just named. Try using your revised shape.

Restoring a Shapes Page

You can change any or all of the shapes on the Shapes page to whatever you want. However, you must remember that when you replace a shape with the new one, the old one is lost. Think carefully before you press Esc after changing shapes!

If you accidentally destroy the shapes that came with your version of LogoWriter, you can restore them using another LogoWriter disk with the correct Shapes page. The directions on how to do this are given below. Be *very* careful when you copy an entire Shapes page; it is easy to accidentally destroy the Shapes page you want to keep.

First, be sure you are on the Contents page.

1. Put the disk with the correct Shapes page in the disk drive.
2. Select the Shapes page.
3. Remove the disk with the correct Shapes page.
4. Put the disk onto which you want to copy the correct Shapes page in the disk drive. (This is the one with the "wrong" Shapes page on it.)
5. Press Esc.

Both disks now contain identical Shapes pages.

Frequently Asked Questions

1. Can I make a shape larger than the shape grid shown on the screen?

   Answer: No. However, you can make up several different shapes and stamp them next to each other. This produces the appearance of a larger shape.

2. Can I start with a shape such as the helicopter, and easily modify it?

   Answer: Yes. Here is how to do it:

   1. Go to the Flip side of the Shapes page.
   2. Go to the shape you want to modify.
   3. Press the Copy Keys (see the Appendix for the correct keys to use with your machine).
   4. Go to a blank shape.
   5. Press the Paste Keys (see the Appendix for the correct keys to use with your machine).
   6. Now modify this shape as desired.

3. Is there an easy way to get rid of a shape I no longer want to use?

   Answer: Yes: Here's how:

   1. Go to the Flip side of the Shapes page.

   2. Go to the shape you want to get rid of.

   3. Press the Cut Keys (see the Appendix for the correct keys to use with your machine).

Bugs and Debugging: Potential Problems When Saving Shapes

Most people are not good at reading a complex set of instructions and following them without making a single mistake. (It is very difficult to write directions so that people can read and follow them without error.) In many computer programming situations the results of an error are not disastrous. You write some instructions and produce an incorrect result. So, what's the big deal? You merely detect the errors and correct them.

This chapter gives you a major new capability–the ability to develop your own shapes. Along with this comes the possibility of making mistakes that will have far reaching consequences, mistakes that are not so easy to correct. You have learned a way in which you can really make a mess of things. For example, you can completely destroy the Shapes page on your LogoWriter Scrapbook disk. As you attempt to correct this, you can make a mess of the Shapes page on someone else's LogoWriter Scrapbook or Master disk.

This means that you want to be very careful as you change your Shapes page. If you do mess this up, your next "line of defense" is copying someone else's Shape page. In this process, if you make a mistake, you may mess up their shape page. At some stage you may want to seek help from a more experienced Logo user. This is a key idea in debugging. An outside expert may be very helpful, and may indeed be necessary. You want to come to understand your own capabilities and limitations. You want to become self-sufficient. However, at times you will need outside help.

Computers and Problem Solving: Building on Your Own Work

In the previous chapter we talked about learning to build on the work of others and how this is a key idea in problem solving. Computers are very important in problem solving because they are a unique new aid to building on the previous work others have done. Often a computer can save you a lot of learning time as well as a lot of work in actually solving a problem.

Another very important idea in problem solving is building on your own previous work. You do this all the time. It took you a great deal of time and effort to learn how to read. Now you read with little effort–building on your previous learning. When you are writing a term paper, perhaps you begin by going to the library and making a set of note cards containing relevant information. Later, as you actually write the paper, you build on the work you have previously done and "stored" on these note cards.

Computers bring a new dimension to the idea of building on your own earlier work. When you save and later reuse a Logo page, you are building on your previous work. When you develop a new shape and later use it, you are building on your previous work. In both cases the computer contributes significantly to your ability to use your earlier work. It stores the results of your work in a form that is convenient for reuse.

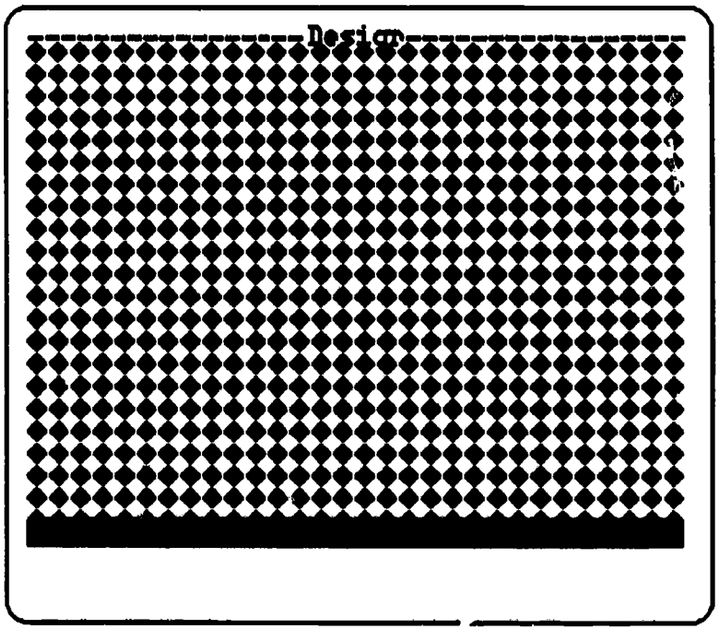
Now you can begin to see one reason why many people want to learn how to write computer programs. Many of the problems you encounter are unique to your interests, abilities, knowledge, and experience. Nobody else has ever viewed the world in quite the way that you view it. Some of the "personally unique" problems you encounter can be solved by computer. However, it is likely that no one else has encountered the problems in just the same way you have. Thus, it may well be that nobody has ever written a computer program to solve these problems.

A computer provides a unique new way of storing the procedures and information needed to solve certain types of problems. This obviously is important in education. Procedures can be memorized and practiced until you can carry them out rapidly and accurately by hand. However, this can take a great deal of time and may well take abilities you do not have. (Do you have the hand-eye coordination to be a "by-hand" graphic artist?) Quite a bit of our current educational process is oriented toward teaching students to master by-hand procedures that computers can also carry out. An alternative would be to place much more emphasis on helping students learn to develop new procedures and to become more effective in solving problems using procedures they have learned or developed. This type of procedural thinking is a new idea in education, and it is gradually gaining strong acceptance as schools place more emphasis on students gaining higher-order thinking skills.

Activities

1. Now that you can create any shapes you want, try a project from the previous chapter but use your own shapes. For example, you can make

   - Signs made up of special symbols.
   - Wallpaper-like patterns.



   - A map showing locations of interest.
   - Cartoons.

As your are developing your project, don't forget to use your journal to document your thoughts, successes, and frustrations.

2. In an earlier chapter you learned how to use RANDOM. Can you use RANDOM to pick a shape? How about picking a random shape, using a random color on it, and placing it on a random-colored background?

3. You can produce animation by using a shape, waiting for a short time, switching to a slightly changed shape, and then repeating the process. For example, you might type

```
REPEAT [SETSH 1 WAIT 2 SETSH 2 WAIT 2]
```

   Try it. How would you make the animation appear to move? Document your experiments in your journal.

4. Think about the computer's ability to provide some people with skills they are not necessarily able to do "by hand." Do you see the computer as a tool that can aid you in some way? How does this new tool affect society? How should it affect education? Jot down your thoughts and observations in your journal.

Chapter 8
FILL and SHADE

Would you like to make a "solid" rectangle that is all one color? This is easy in LogoWriter. With LogoWriter you can fill in areas on the page with the current pen color, and you can also fill areas of the page with turtle shapes—even with shapes you have created!

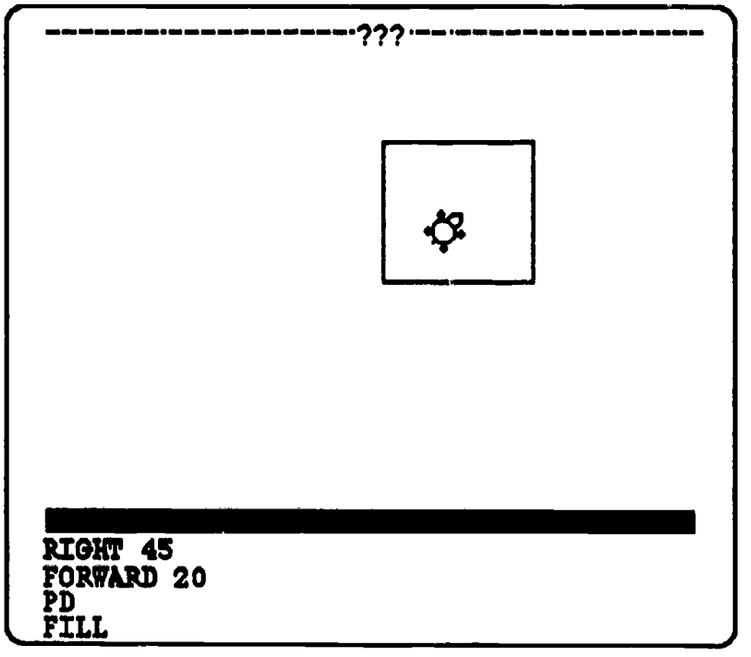
Clear your page and type

```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

to draw a square on the screen. Next type

```
PU
RIGHT 45
FORWARD 20
PD
```

The turtle is now inside the square!



Now type

```
FILL
```

The square fills with color.

Type

```
CG
SETC 5
```

Use the arrow keys to move up to the REPEAT line you used to draw the square. Press Return/Enter to run each line below the REPEAT. Now the turtle uses the color corresponding to SETC 5 to draw a colored square and fill it with color.

Notice that the pen was lifted before moving the turtle into the square. If you leave the pen down, move the turtle and then type FILL, nothing happens. This is because the turtle "senses" that it is already on top of a colored spot. Always be sure to lift the pen before moving the turtle into an area to be filled.

You can fill any closed area with whatever color you want. Be careful, however, that the area really is closed. Otherwise the color may "leak out" all over the page! Try this. Make a figure that is not quite closed,

```
CG
REPEAT 3 [FORWARD 50 RIGHT 90]
FORWARD 45
```

move the turtle "inside" this figure, and then type FILL. What happens?

Occasionally FILL does not work as you expect when you are quite sure you have the turtle placed inside a closed figure and you typed FILL. In this case, move the turtle just a few turtle-steps and then try again. Remember, keep the pen up when moving the turtle; put the pen down when you are ready to fill.

Next, make another square with the turtle inside.

```
CG
REPEAT 4 [FORWARD 80 RIGHT 90]
PU
RIGHT 45
FORWARD 20
PD
```

Now type

```
SETSH 29
```

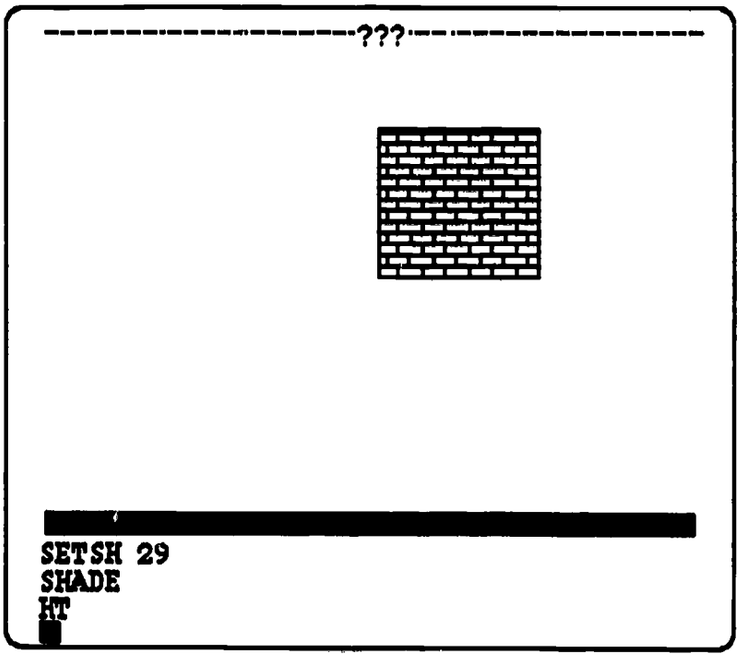
(Here we are assuming you are using the Intermediate version of LogoWriter and have a shape of small bricks. If you are using the Primary version of LogoWriter, use shape 19.)

Then type

```
SHADE
```

and finally

```
HT
```



Now you see a square filled with bricks. Use the arrow keys to draw the square again and repeat this sequence, but use a different turtle shape. The square will fill with whatever shape you select. Try filling a figure with kittens.

You need to remember to use the same steps when working with SHADE that you use with FILL. Lift the pen when moving the turtle, put the pen down before shading, and move the turtle just a few steps if you have difficulty.

FILL and SHADE can be used to create interesting effects in drawings. Spend some time experimenting with them. Keep track of your work in your journal. What was particularly interesting? What ideas might you use in the future? Which of your results make you think of other neat projects?

Frequently Asked Questions

1. Sometimes FILL and SHADE don't work. Is there something I am doing wrong?

   Answer: If either FILL or SHADE don't work properly, it is most likely because you don't have the pen down or the turtle's pen is over a dot that is already filled or shaded. First, type PD and try again. If that doesn't work, then the turtle is positioned so that LogoWriter thinks the area is already filled or shaded. Lift the pen. Move the turtle a small distance. Put the pen down and try again.

2. I messed up a very complex picture when I used FILL. It took me a long time to redo the picture. How can I avoid this problem?

   Answer: Since shading and filling areas make major changes on your page, it is a good idea to save your page *before* you type SHADE or FILL. Suppose your page is named MY.PAGE. First save the page and then get it from the Contents page again. Then do your filling or shading. If the FILL doesn't work right, rename the page. For example, you might type

   ```
   NP "JUNK
   ```

Press Esc. This saves the page "JUNK." Then get the page MY.PAGE from the Contents page and try again.

3. Do FILL and SHADE work exactly the same in all versions of LogoWriter?

   Answer: No. The kinds of problems you might encounter with FILL or SHADE may differ from one brand of computer to another, just as the colors available differ from from one kind of computer to another.

Bugs and Debugging: Is Logo Wrong?

As you work with LogoWriter—or any very complex computer software—you may get some unexpected results where you are absolutely positive you have not done anything wrong. It is possible you have encountered a bug in the software you are using.

LogoWriter itself is a very complex program. Earlier in this chapter, we indicated that FILL occasionally doesn't work correctly. The FILL command is a small part of LogoWriter, but it is an unexpectedly complex part. A rule of thumb is that every complex computer program contains bugs. When a complex program is being written, a great deal of effort goes into detecting and correcting bugs. On such a very large project there may be a number of people hired just to test the program, looking for bugs. The program may be sent out to a number of test sites so that a variety of people who may want to purchase the program can try it out and look for bugs.

Even with all this testing and debugging, there will be undetected bugs, or bugs that were detected but were not considered important enough to fix. Thus, when you use a complex program such as LogoWriter, it is possible that you will discover an error in the program. However, don't be too quick to jump to the conclusion that you have discovered a bug in LogoWriter just because you are not able to explain an incorrect result in any other way. It is not uncommon for beginners to assume that the problems they encounter are problems in the hardware or software. You need to be aware that it is much more likely that the bug is yours rather than LogoWriter's.

If you discover a supposed bug in LogoWriter, you will want to carefully document your find. That is, you will want to write detailed notes in your journal that tell exactly how to make the bug reappear. Then you can share the bug with your course instructor or with the company that created LogoWriter. Some bugs are so complex that you cannot readily make them reappear. In such a case you may never know whether the bug is in LogoWriter or due to something you did wrong.

Computers and Problem Solving: "I Can't Do Logo."

Many students experience considerable difficulty in dealing with FILL and SHADE at this stage of using this book. It isn't that these two primitives are particularly difficult. Rather, it's that these two new primitives tend to be the straw that breaks the camel's back.

This is an important idea in problem solving and in dealing with learning. It is easy to learn one new thing, such as one new primitive. You learn it and you use it. Not much can go wrong. Then you learn a second primitive, a third primitive, and so on. Several difficulties begin to arise:

1. You begin to get new primitives confused with ones you have used previously. You make small mistakes in using a primitive learned on a previous day, perhaps because something about it is nearly the same as a part of a new primitive.

2. You begin to try to solve more complex problems. These require that a whole sequence of instructions be specified. However, you have not yet learned how to effectively develop a long

sequence of instructions, and the methods that you make up for yourself prove inadequate. Eventually you attempt problems that are just too hard for your current level of knowledge and skill.

3. You have not developed appropriate skills for determining what you are doing wrong and how to correct your mistakes.

Think about learning to add, subtract, multiply, and divide whole numbers, fractions, decimal fractions, and negative numbers. Do the above three ideas apply there? For many students they do! The result is often a feeling of "I can't do math" and a withdrawal from further attempts to learn mathematics.

Let's assume you have now reached a stage in your study of Logo where you are frequently getting stuck and are spending a lot of time in rather fruitless trial and error. Great! You have now encountered a very valuable learning environment. What do you do when you can't figure out what to do? (Be aware that some students at this stage say, "I can't do Logo programming" and withdraw from further attempts to learn it.) This is an excellent topic to write about in your journal. What roles can a book or a teacher play in helping a student deal with this type of frustrating situation? What does it mean when a person says, "I can't do math" or "I can't do Logo?"

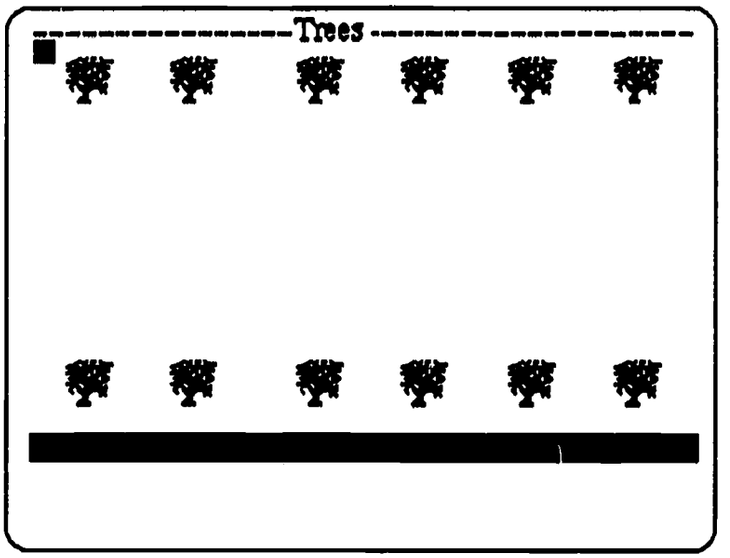
Activities

1. Create a number of closed polygons on the screen. Use Turtle Move to place the turtle before drawing each design. Then again use Turtle Move to get the turtle inside each design to fill it. Can you create some interesting artistic effects using filled areas? What happens when the polygons overlap? What kinds of problems did you encounter? How did you solve them? Keep a record in your journal.

2. Use SHADE to create patterns that are part of a greeting card. For example, you might place lines so that the edges of the page are filled with a shape of your choice. You might even want to design a turtle shape of your own to personalize the card.

3. Design the graphics for a sign announcing some school or community activity. Use a combination of filling, shading, and turtle moves to make your sign.

4. Use the LogoWriter techniques you have learned so far to create the cover for a report. You could create a turtle shape appropriate to the subject of the report.



5. Address the question "What do you do when you can't figure out what to do?" in your journal. How do you solve such problems when working with Logo? How do you solve such problems in other domains? As a teacher, how can you help your students learn to work with such "unsolvable" problems?

Chapter 9
Mixing Text and Graphics

So far you have been primarily using graphics on the LogoWriter page. The text in the Command Center at the bottom of the screen is not part of the page. It is not saved when you save the page. In this chapter you will learn more about putting letters and numbers on the page so that they can be saved along with the graphics when you press Esc.

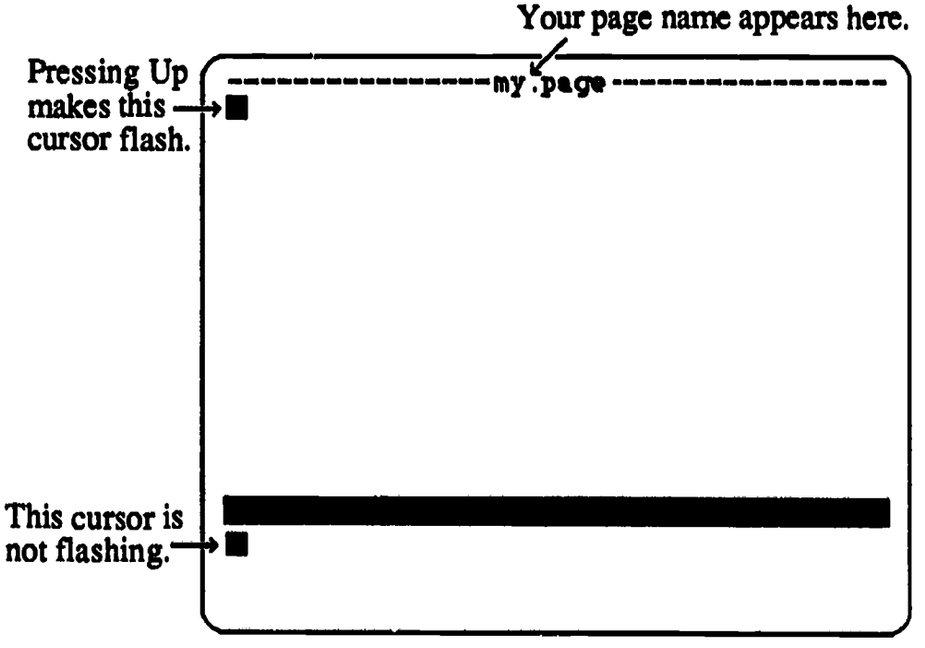
Start LogoWriter and get a new page. Name your page by typing

```
NAMEPAGE "page.name.you.choose
```

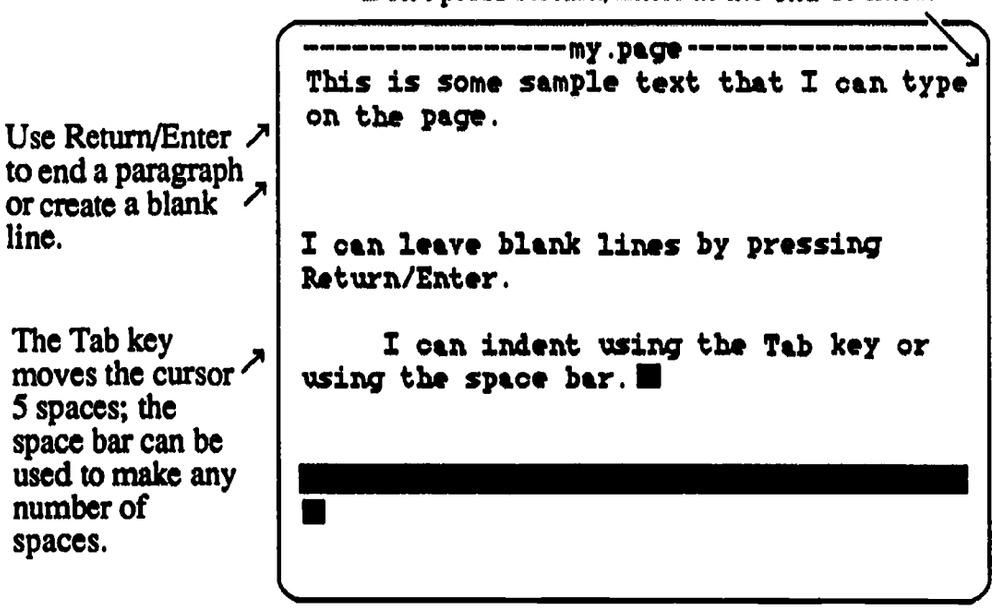
Hide the turtle by typing

```
HT
```

and then press the Up keys. (Check the Appendix for the keys to use for your particular machine.) Notice that the cursor is now flashing at the top of the page. You are now in a word processor mode. As you type, you will be using the word processor that is built into LogoWriter. It works much like an ordinary word processor.



Begin typing. The words appear on the Page instead of in the Command Center. Continue typing. Do not press Return/Enter when you get to the end of a line.



Complete words automatically jump to the next line. This is called *word wrap* and is a standard feature in word processors.

Now that you have some text on the page, press the Down keys (again check the Appendix) to activate the cursor in the Command Center. Next type

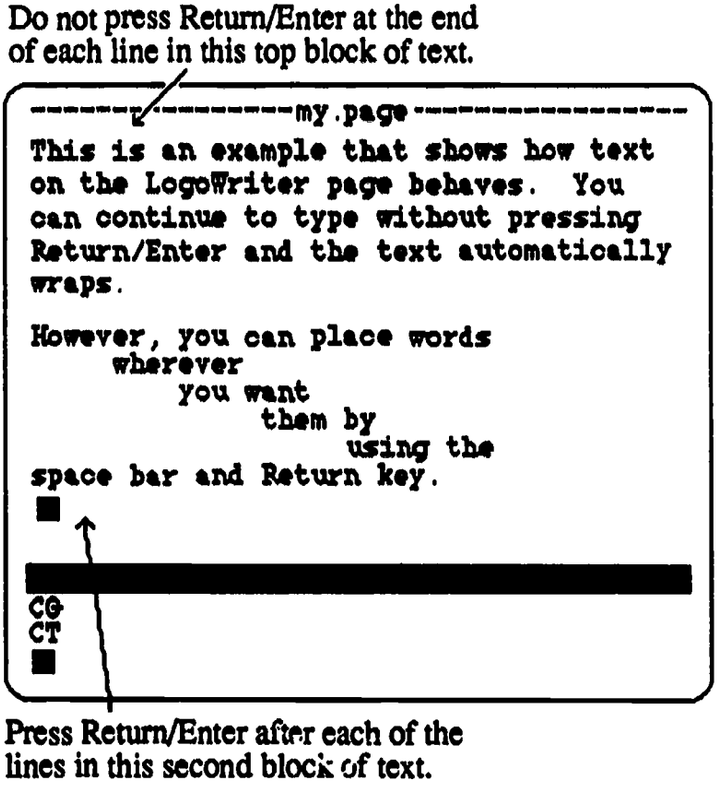
```
CG
```

Nothing happens. That is because the words you have typed onto the page are in text mode instead of graphics mode. To clear the text from the page, you must type

```
CT
```

for Clear Text.

To explore how the LogoWriter word processor works, clear the page (CT), hide the turtle (HT), press the Up keys, and type the following text as shown. Use the space bar and Tab key to indent the lines in the second paragraph.



Printing Text

You already know that PRINTSCREEN will print the page on the printer. This works with graphics, text, or a combination of the two on a single page.

Sometimes you will want to print just the text you put on the page using LogoWriter just as a word processor. Type

```
PRINTTEXT
```

The words appear on the printer in the same places that they are on the page and in regular sized type. In addition, there are margins at the left, top, and bottom of the page. If you want double spacing, you can use DSPACE. Use SSPACE to get back to single spacing.

Finally, type

```
PRINTTEXT80
```

and the text is printed using 60 columns (unless you pressed Return/Enter after each line) with margins at the left, right, top, and bottom of the page.

Take a few moments to be sure you understand how each of these commands for printing text works. Note that PRINTSCREEN prints all the text *and* graphics that appear on the page. PRINTTEXT and PRINTTEXT80 print only text.

Note: This book assumes that your instructor will make you aware of any necessary details for using the equipment available to you for printing. However, there are a few differences among versions of LogoWriter you may need to be aware of.

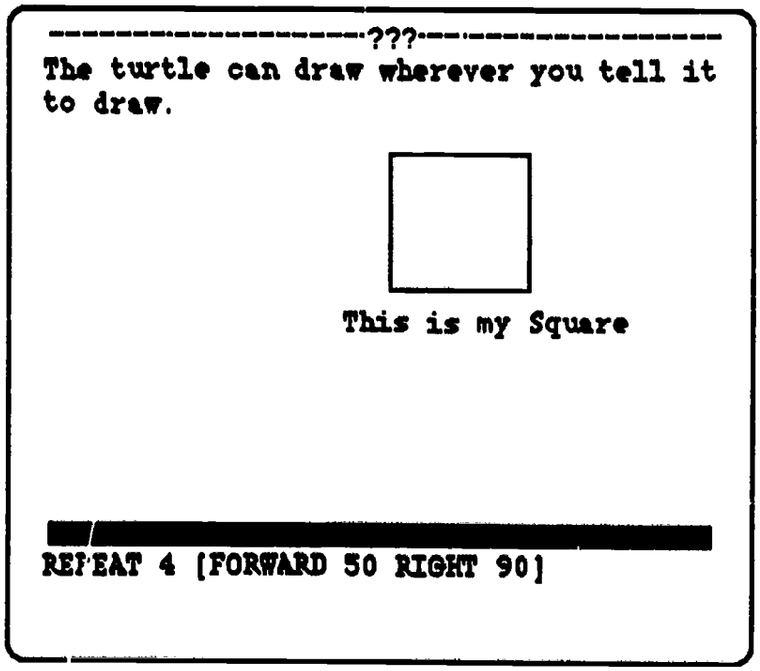
- If you are using LogoWriter version 1.0, there are no margins on the page.
- If you are using LogoWriter version 1.1, you will need to type an SSPACE before typing PRINTTEXT.
- If you are using IBM Version 1.1, you must use SINGLESPACE instead of SSPACE; DOUBLESPACE instead of DSPACE.
- PRINTTEXT80 is set up for 10 characters per inch (Pica type). If your printer is set for some other type style, the text may not go all the way across the page. Sometimes turning your printer off and then on again will reset your printer to 10 characters per inch.

Combining Graphics and Text

You have now learned how to put text on the page, and in previous chapters you learned how to put graphics on the page. Using LogoWriter, you can combine graphics and text on the same page. Get a new page and try this:

```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

Now use the Up keys to get into text mode. Type "The turtle can draw wherever you tell it to draw." Then use the Return/Enter key, the Tab key and the Space bar to put "This is my Square" on the page so that it looks approximately like this:



Next, press Esc to save the page. Get the page and again type

```
CG
```

What happens? Do you see why?

Now type

```
CT
```

What happens? Do you see why?

When you are working on a project that contains both text and graphics, you can correct the parts separately or even start one part over without erasing the other. This can be very handy when working on a more complex project.

Frequently Asked Questions

1. Is the LogoWriter word processor a "full-featured" word processor?

   Answer: No. If you have experience with other word processors, you may wonder how you can accomplish such tasks as centering text, setting margins, or underlining words. None of these features is available in the LogoWriter word processor. However, the LogoWriter word processor has its own advantages. It is very easy to use. There are no elaborate menus or key combinations to learn. In addition, you will soon learn that you can easily insert graphics into your text. If you want to do simple desktop publishing, then LogoWriter is an excellent tool. If you want to write long papers, you probably want to use another word processor. Don't try to make the LogoWriter word processor meet all your word processing needs.

2. I accidentally pressed CT and it erased everything I had typed. Can I get it back?

   Answer: If you type CT, all the text on the page disappears. If typing CT was an accident, it is possible to get your text back. *Before you press any other keys*, type

   ```
   UNDO
   ```

   in the Command Center and your text will reappear. (Note: UNDO is not available in LogoWriter Versions 1.0 and 1.1).

3. What's the best way to avoid the frustration of a major error or a power failure wiping out a lot of work?

   Answer: Make it a habit to save your work every 10 - 15 minutes. Better yet, save it every time you complete a major change on your page. The time taken to save your work will pay off later if you make a major mistake or there is a power failure.

4. Can I put text so that it is on top of a picture?

   Answer: This is the type of question you should answer for yourself. Try it, and see what happens.

Bugs and Debugging: Dealing With Complexity

The original versions of Logo did not contain a word processor. In most early versions of Logo for microcomputers, you could only put text at the bottom of the screen. If you wanted words with your pictures, you had to have the turtle draw each letter–a tedious task at best. With these early versions it was certainly not convenient to use Logo to write a story and illustrate it with graphics.

The word processor in LogoWriter adds still another dimension to the level of complexity you face as you learn the language. In addition, it allows you to attack still more complex problems. Thus, you may be experiencing more and more frustration as you work with Logo. If so, here are a few suggestions:

1. Make a list of the primitives that have been covered so far. In your list, include an example of correct use of the primitive and/or a short explanation of how it works. Keep this list at hand as you write programs.

2. Practice creating new materials in the context of older materials that you are certain you understand. For example, suppose that you are quite confident of your ability to draw a square, but you are not too sure how to put text exactly where you want it on a page. Draw a square (a familiar task) and practice putting text at various locations relative to the square. Similarly, if you are not sure how the FILL and SHADE primitives work, practice them using a simple square.

3. Have you found yourself "thrashing around" when Logo won't seem to do what you want? Do you get upset? Angry? Do you begin to enter commands randomly? Are you debugging only by trial and error, not really knowing what you are doing? If so, then do some deep breathing exercises, get up and walk around for awhile, or do something else to help you to overcome the panic or upset. View the situation as an important learning experience.

Computers and Problem Solving: Problem Solving Styles

To solve problems, a person needs to have an appropriate combination of lower-order skills, higher-order skills, knowledge of the problem field, perseverance, and so on. As you learn Logo, you can examine what works best for you. Are you really good at learning the primitives and writing isolated instructions? Or does this bore you? Are you good at envisioning a complex Logo drawing task and carrying it out, or do you feel overwhelmed by such a task? Do you have good persistence, or are you easily distracted?

The Logo environment provides a good place for self-examination and/or for research into problem solving. How do children learn to solve problems in a Logo environment? Is this the same way adults learn? Do all children learn in the same way? What role should the teacher play when working with children in a Logo problem solving environment? What should students be expected to discover for themselves, what should they teach each other, and what ideas should the teacher suggest? These are all questions that can be researched. Answers gained in a Logo research environment may give us insights into teaching in other environments.

The teacher needs to decide the goals for having students work in a Logo environment. Suppose the goal is to learn to solve multistep Logo problems. Suppose that essentially all the instruction and all the practice exercises are lower-order types of activities involving single steps or just a few steps. Will this type of instruction and practice help adequately prepare you to solve more complex problems? Is this the most effective approach, and is it equally effective for all learners?

As you think about this, be aware that many people argue that it is essential to master the "basics" before proceeding with more difficult tasks. However, there are strong arguments on the other side, and educational research is lending credence to these arguments. Many schools now use a combination of process writing and whole language experience to teach writing. They do not get the students bogged down on details of spelling and grammar as they are working to express ideas in writing. Similarly, process math and process science are increasingly gaining acceptance. Many Logo leaders believe in using a process-oriented, discovery-based approach to helping students

learn Logo. They argue strongly against the approach that it is necessary to master the basics before beginning to work on harder problems.

Activities

1. Now that you know how to place both text and pictures on the screen, you can use this capability to do a variety of things.

   - Make a sign or a poster (printed using PRINTSCREEN).
   - Write a friend a letter describing a picture you drew using the turtle, and include the picture.
   - Create a greeting card or invitation.

   - Do a short report for another class, combining text and pictures.
   - Write a poem and illustrate it with pictures.
   - Make a design or picture using letters, numbers, and other keyboard symbols. (Pictures such as this used to be called "typewriter pictures.")
   - Create new shapes and use them to fill areas with interesting designs or patterns.

   As your work on your project, continue to examine your own thinking and your own problem solving skills. Keep a record in your journal.

2. Write in your journal about how you feel on the need to master the basics before proceeding to solving problems requiring higher-order cognitive skills. Focus specifically on your own learning of Logo and your knowledge of how young students learn.

3. Compare and contrast process writing with "process Logo." Do some subjects lend themselves better than others to the process approach in learning/doing?

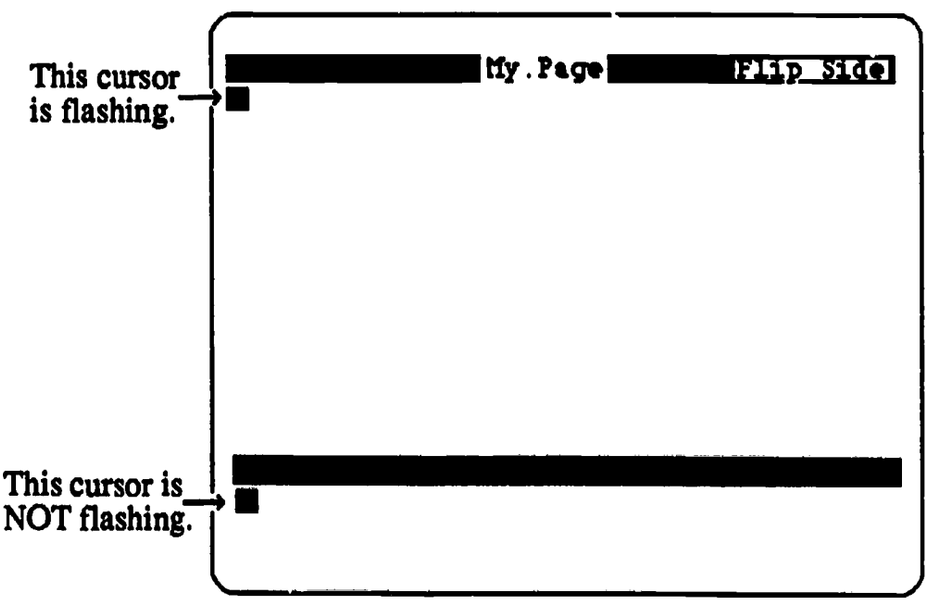
4. Think about the frustrations you have experienced in your work with Logo? Look over your journal entries to see how you have dealt with those frustrations. How can you transfer what you have learned so far about dealing with the frustration of solving hard problems to solving hard problems you will encounter in the future? How can you teach your students to deal with frustration in a problem solving environment?

Chapter 10
Writing Procedures

All the pages you have created so far have been done by typing instructions in the Command Center. When you saved, everything you created was put on your disk exactly as it appeared on the page. This is called programming in *Immediate Mode*. The computer does exactly what you tell it to do immediately after you issue the instruction. In this chapter you will learn about a different mode–one in which you can type a long sequence of instructions and then have the computer carry out these instructions.

Have you ever created a neat graphics design that you would like to use as part of another page? Perhaps you remembered the instructions you used to make it, perhaps not. Without a list of the instructions you used, it can be hard to recreate a graphics design. You have already seen that the instructions in the Command Center are not saved. If you start LogoWriter at another time and get your page, the instructions you used to create it are not there.

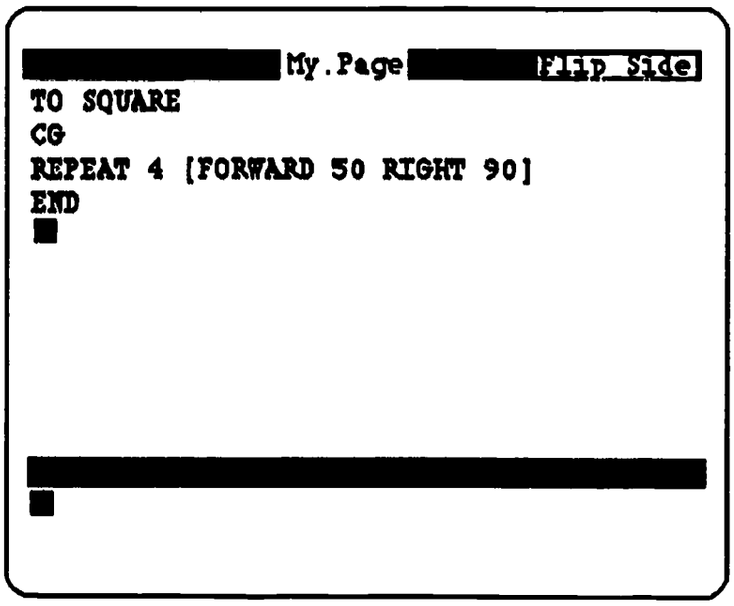
There is a way to save instructions that you have used to create pages in Logo. Recall that you could flip the Shapes page to see magnified copies of the shapes. You can also flip the page on which you have put graphics. Try it. Use the Flip keys (see the Appendix) to see the back of a page containing a graphics design. Notice that it says "Flip Side" at the top of the screen and that the Command Center is still visible. Use Flip to go back to the front of the page. Notice the location of the flashing cursor. Where is it when you are on the front of the page? Where is it when you are on the back of the page?



Now try this. Start on the *front* of the page. Flip the page and use the Down keys to get to the Command Center. Notice that the cursor is flashing in the Command Center. Use the Up keys. What happens? Practice using the Down and Up keys while you are on the Flip side of the page, so that you are confident that you can move between the cursor in the Command Center and the cursor on the Flip side of the page.

63

With the cursor flashing at the top of the Flip side of the page, type the lines shown below. (Remember that you can correct keyboarding errors and can move the cursor around on the screen using the same keys you use to move around in the Command Center.)



Flip to the front of the page and type

```
SQUARE
```

What happens? You have just written and run your first Logo procedure!

A *procedure* is a collection of Logo commands. Procedures must begin with the word TO and end with the word END. The structure of a procedure is as follows:

```
TO name.of.procedure     <--title line
Instructions go here     <--body of the procedure
END
```

Procedures must be written on the Flip side of the page. They are then (automatically) saved with the page.

Continue the above activity by typing

```
CG
```

so the square is erased, and then save your page. Now select the page that you just saved, and notice that there is no picture of a square on the page. Type

```
SQUARE
```

What happens? You have just made use of a procedure that you have saved!

While the mechanics of procedure writing are rather simple, the idea embodied in procedure writing is profound. When you create a series of instructions to complete a particular task and then put them into a named procedure, you have "taught" Logo a new word. This new word behaves exactly as if it were a primitive. If you don't look on the Flip side of the page, you have no way of knowing whether the instructions you type in the Command Center consist of primitives or user-defined procedures. The fact that you can easily create procedures that function exactly like primitives is what makes Logo an easily *extensible* language.

The Flip side of the LogoWriter page can contain as many procedures as the memory of the computer and/or the version of LogoWriter you are using allows. Each of these procedures must have a unique name. Each new name becomes part of the LogoWriter vocabulary available when you use the page. All of the primitives you have learned so far can be used in writing procedures. Here are a few examples of procedures you may find quite useful.

You can move the turtle around.

```
TO PLACE.TURTLE
CG
PU
FORWARD 20
RIGHT 60
FORWARD 30
PD
END
```

You can change the shape of the turtle and the color of its pen.

```
TO CHANGE.TURTLE
ST
SETSH 25
SETC 3
END
```

You can stamp turtle shapes.

```
TO STAMP.IT
SETSH 15
PD
STAMP
PU
FORWARD 20
END
```

You can even put both text and graphics on the page.

```
TO LABEL.A.SQUARE
REPEAT 4 [FORWARD 60 RIGHT 90]
PRINT [This is my square.]
END
```

Take some time now to experiment with writing a procedure. Notice as you experiment how powerful this idea of procedure writing is. Now *you* are teaching Logo how to do new things. The computer is now under your control.

Try writing and running a procedure, and then changing the procedure and running it again. You edit a procedure just the way you edit any other text. You can insert and delete lines, or make corrections, exactly as you do in the Command Center or as you did in the previous chapter when you were using the word processor. If you want to erase all the procedures on the Flip side of the page, you can go to the Command Center (using the Down keys) and type CT. The procedures will disappear.

Two Kinds of Computer Programming

Computer programming "tells a computer what to do" using a general purpose programming language such as Logo. However, you have now seen two different forms of computer programming. Prior to this chapter, each instruction you typed was carried out immediately. In this chapter, you have learned to write a sequence of instructions (a procedure) that is not carried out until a later time. We talk about programming in the Immediate Mode and programming using procedures.

Some programming languages, such as early versions of Pascal for microcomputers, only allow procedural programming. Other programming environments, such as simple spreadsheets, only allow you to give instructions one at a time, in immediate mode. Logo allows both. Thus, you must decide which to use when working to solve a particular problem. Here are some rough guidelines:

1. If the task to be accomplished requires only a small number of steps and you only need to carry out the task once, use the Immediate Mode.

2. If the task to be accomplished requires a large number of steps, write a procedure or procedures.

3. If the task to be accomplished needs to be done more than once, write a procedure or procedures.

Frequently Asked Questions

1. I'm confused by all of the different "sides" and cursor positions. How can I tell where I am as I move among different LogoWriter features?

   Answer: When you begin using the Flip side of the page to write procedures, it is easy to get lost. Learn to look at the screen carefully.

   - Are you on the front of the page or the Flip side? (You see "Flip Side" if you are not on the front.)

   - Where is the cursor flashing?
     Not at all? You are in Turtle Move mode on the front of the page.
     At the bottom of the screen? You are in the Command Center.
     At the top of the screen, no Flip Side? You are in the word processor.
     At the top of the screen and it says Flip Side? You are ready to write procedures.

2. My procedures disappeared? What did I do wrong?

   Answer: Earlier you learned that the Command Center is visible on both sides of the page. Now you have discovered that the same text editor is used on both sides of the page. The CT

(clear text) command erases all the text on *whichever side of the page that is visible* when the command is typed. A common way that procedures get lost is the following:
Use CT to clear the front of the page
Flip the page.
Move the cursor to the Command Center
Use the arrow keys to move the cursor to the CT command.
Press Return/Enter

Usually this happens by accident. You must be careful that the Flip side of the page is *not* visible when you use CT. (Remember, if you accidentally erase text on *either* side of the page, you can type UNDO immediately to restore that text.)

3. Sometimes I don't want to erase all of the procedures on the page. Removing a procedure one line at a time is tedious. Is there an easier way?

   Answer: You can erase one *line* of text by using the Erase to End of Line keys (see the Appendix). This key combination removes characters from the current location of the cursor to the next place you pressed the Return/Enter key.

   Incidentally, the Erase to End of Line key combination can be used on the Contents page to erase a page you no longer want. Place the cursor on the page to be erased from the Contents page and press the Erase to End of Line key combination. Be careful, though. Once you erase a page from the Contents list, there is no way to get it back again.

4. I can see my procedure on the Flip side of the page, but when I type the name of the procedure, Logo responds with "I don't know how to..." Why won't my procedure work?

   Answer: First, check to be sure that you have spelled the name of the procedure correctly. If you accidentally name your procedure SQAURE and then type SQUARE in the Command Center, Logo responds with the "I don't know how to..." message since it only understands the procedure "SQAURE."

   Each procedure must begin with TO and end with END. If you have two procedures on the Flip side of the page and you omit the END statement on the first one, then Logo will not see the second procedure. It is a good idea to leave a blank line between the END of one procedure and the title line of the next. This helps you avoid errors such as missing END instructions.

Bugs and Debugging: Procedures and Playing Turtle

Procedure Mode programming is much more likely to cause difficulties than Immediate Mode programming because it allows you to attack much more difficult problems and because you do not immediately see the results of each instruction you write. Instead, you must imagine the results of these steps in your mind's eye.

Debugging a procedure begins at the time you first start thinking of creating a procedure. You imagine in your mind's eye what you want the procedure to accomplish. Perhaps you sketch a drawing of the result you want. Perhaps you write the instructions by hand before entering them into the machine.

Next, you write the procedure and run it. Your procedure may contain one or more syntax errors and one or more errors in logic. All errors need to be detected and corrected in order for the procedure to produce the results you want. You will find that you frequently end up reading and rereading a procedure, looking for errors.

Initially most people find it difficult to read a procedure and to "see" in their mind's eye what the procedure does. They can read the words and understand the meaning of each individual instruction. However, they do not readily comprehend the overall "meaning" of the sequence of instructions in the procedure.

It is very important to learn to read a procedure and *hand trace* it–that is, carry out the steps in your head and/or with use of pencil and paper. This is a standard technique in debugging procedures. You will get better at it with practice. One way to practice is to explain a procedure to a friend. That is, if you have a procedure that is producing incorrect results, carefully explain each instruction line in the procedure to a friend. This type of talking out loud to yourself or to someone else will often help you to more fully understand a procedure. Indeed, it will often help you to discover your own bugs. Often your problems with Logo occur because the turtle is not moving the way you think you told it to move. A form of hand tracing in Logo is called "playing turtle." To play turtle, you physically walk through the steps that you told the turtle to take. This can be very helpful if done with a friend and is particularly powerful when used with children.

Computers and Problem Solving: Procedures as Building Blocks

Each primitive in a programming language is a building block. The building blocks in a particular programming language are chosen to help solve certain kinds of problems. For example, Logo includes a number of graphics primitives useful in drawing pictures. COBOL contains primitives useful in solving business problems. BASIC contains primitives useful in solving math problems. PASCAL was specifically designed to help computer science students learn some of the most important ideas of computer programming.

Logo is an extensible language. This means that it contains particularly good provisions for adding commands and reporters that are needed. You can personalize Logo to be particularly useful in solving the types of problems that are of most interest to you. In essence, you can create your own personal programming language by writing appropriate procedures to fit your specific needs.

This adds a new dimension to problem solving. Think about some overall problem solving task, such as designing a house. What are the commands that would be most useful? It is clear that it takes a great deal of understanding of architectural design to answer this question. That is, domain specificity is an issue in the design of your own programming language. If you don't know much about architectural design, it is not likely that you will figure out what to add to Logo to help you solve architectural design problems.

This analysis suggests a major trend in education that will emerge during the next decade. Within each academic discipline a set of computer tools is being developed specifically to meet the needs of professionals in the discipline. This developmental work is being done by a combination of professionals in each field and professional computer programmers. Eventually, instruction in the use of a discipline's computer tools will become a routine part of instruction within that discipline. We already see this in fields such as architecture, business, graphic arts, and publishing.

Activities

1. Write a procedure to make a small design. Flip to the front side of the page, and then use a combination of Turtle Move and your procedure to place your design in various places on the page. What kind of projects does this activity bring to mind? Write in your journal.

2. Write a procedure to stamp several different shapes on the page.

3. Write a procedure to stamp a short row of shapes on the page. Flip to the front side of the page and use a combination of turtle move commands, REPEAT, Turtle Move mode, and your procedure to make a border of shapes around the edge of the page. What kind of problem solving strategies did you use to complete this activity? Document your thinking in your journal.

4. Write a procedure to make an elaborate graphics design using the turtle. Flip to the front side, use your procedure to make the design, and then use the word processor (press the Up keys) to add a description of your design on the screen.

5. Create a shape of your own and then write a procedure to stamp your shape in a pattern on the page. Add some text.

6. Write a procedure that draws a graphics design and changes the pen color randomly with each line it draws. You might use something like the fluffy ball created in Chapter 4. Put a title on your design.

7. Do some journaling on the idea of domain specificity in problem solving as it relates to developing your own procedures (extensions of LogoWriter) to solve problems of interest to you.

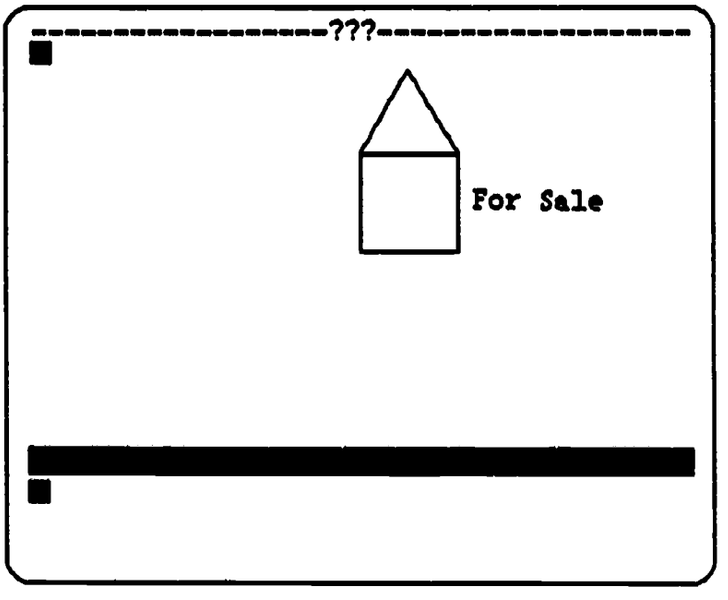
8. How many times have you written the instructions to draw a square in Logo? Perhaps SQUARE should be built in to Logo, at least for your uses. Think of other "tool" procedures you might write that would be useful to you in your work with Logo. Write your thoughts in your journal. Then, perhaps you'll want to begin writing your own tool page that contains useful procedures.

Chapter 11
More Than One Procedure

You now know how to write a procedure using the Flip side of the page. As mentioned in the last chapter, you can write as many procedures as you want on the Flip side of the page, as long as there is space in the memory of the computer. This allows you to create a number of procedures that, when appropriately put together, can solve a complex problem.

Carefully follow the example given below. It shows you how to write a number of procedures and put them together into a rather complex computer program.

Suppose you wanted to have the turtle draw a house on the page and then put a "For Sale" sign next to it. The house is to consist of a square with a triangle on top for the roof.



You can, of course, do this from the Command Center. However, if you do it with procedures, then you can easily make a number of identical houses. Or you can easily make changes to the house.

To begin, you will want to create a procedure for a square and a procedure for a triangle. Flip the page and write these procedures for drawing a square and a triangle.

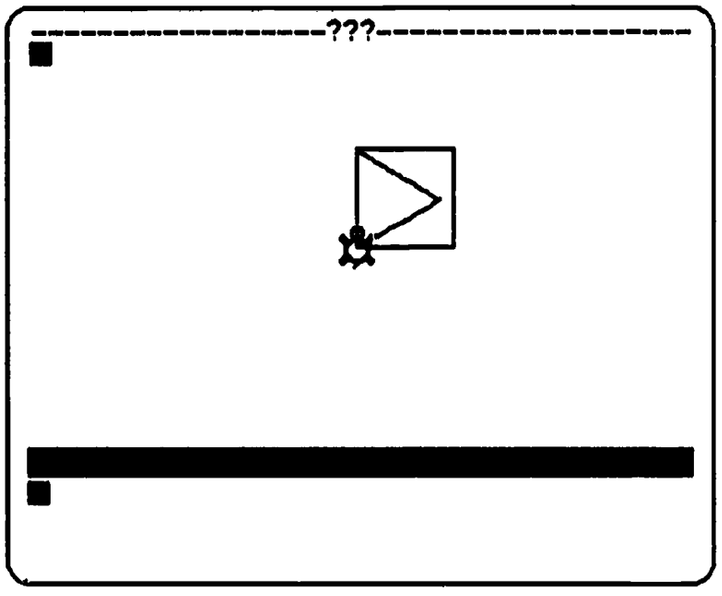
```
TO SQUARE
REPEAT 4 [FORWARD 40 RIGHT 90]
END

TO TRIANGLE
REPEAT 3 [FORWARD 40 RIGHT 120]
END
```

Flip the page back to the front. Type

```
SQUARE
TRIANGLE
```

This draws a square and a triangle. Will the picture look like a house?



You see the parts of the house, but they are not in the right place. The triangle needs to sit on top of the square to make the house. Type CG and see if you can figure out the instructions needed to get the roof in the right place. These instructions should become part of a procedure to get to the roof. Remember to flip the page to enter your procedure.

One solution for placing the roof after having drawn the square might be

```
TO GET.TO.ROOF
FORWARD 40
RIGHT 30
END
```

Now flip the page to the front and type

```
CG
SQUARE
GET.TO.ROOF
TRIANGLE
HT
```

Does the house look the way you want it to? If not, modify the procedures until you are satisfied with your house. These procedures you have created can become part of a HOUSE procedure:

```
TO HOUSE
SET.UP
SQUARE
GET.TO.ROOF
TRIANGLE
END

TO SET.UP
CG
CT
HT
END
```

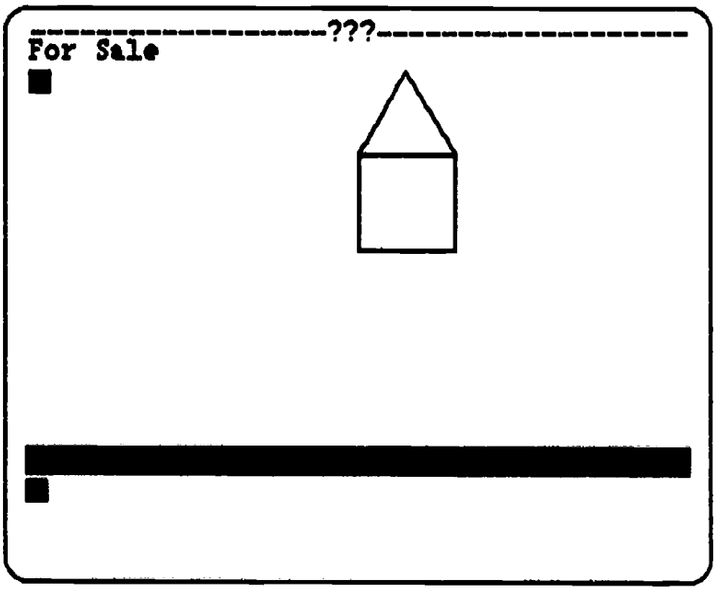
Now, you can type HOUSE from the Command Center and see your house. Notice that a SET.UP procedure was added to prepare the page for the house drawing. Why do you suppose the CT instruction is included in this procedure?

Using PRINT and INSERT

In Chapter 5 you learned that PRINT could be used to print out numbers on the page. It can also be used to print text. Try typing

```
PRINT [For Sale]
```

You see



The text to be printed is in square brackets. After the printing has occurred, the cursor moves to the next line below the text that was printed. Now you need to learn how to position the "For Sale" sign correctly on the page. This takes two steps. First you need to learn to move the cursor down the page using PRINT. Second, you need to learn how to move text to the right on the page.

Begin by typing

```
PRINT []
PRINT []
PRINT []
PRINT [For Sale]
```

You have printed three blank lines above your "For Sale" message. This illustrates how to do vertical spacing of text.

Now you can begin writing your SIGN procedure. Flip the page and type

```
TO SIGN
REPEAT 5 [PRINT []]
PRINT [For Sale]
END
```

Be careful. One set of square brackets are needed after PRINT to get a blank line. Another set of brackets are needed after REPEAT to enclose the list of instructions.

The sign isn't yet where you want it. How can you write a procedure that acts like the space bar? To accomplish that task, you need a new command called INSERT. Flip to the front of the page and try typing

```
CT
INSERT [For Sale]
```

The cursor is after the words "For Sale" and on the same line. When you use PRINT, the cursor always moves to the next line. (Try it!)

The next task is to use INSERT to put blank spaces in front of "For Sale." Try typing

```
INSERT CHAR 32
INSERT CHAR 32
INSERT CHAR 32
```

The cursor moves three spaces to the right. The number 32 is the ASCII code that represents the space bar. (ASCII is a common code used by computers to represent the keys you type on your keyboard.) CHAR is a reporter that returns the character associated with the number to Logo, in this case a blank.

Now you are ready to complete your procedure to make your sign. Go to the Flip side of the page. Type

```
TO SIGN
REPEAT 5 [PRINT []]
REPEAT 28 [INSERT CHAR 32]
PRINT [For Sale]
END
```

With a little trial and error you can position the sign exactly where you want it.

Now try typing

```
HOUSE
SIGN
```

The house appears along with its sign. Now you can put this all together in one procedure. Flip the page, move the cursor to the top of the page, and type

```
TO HOUSE.AND.SIGN
HOUSE
SIGN
END
```

Now, type HOUSE.AND.SIGN in the Command Center and you see the picture that was planned at the beginning of this chapter. You have written a complex set of interconnected procedures.

When you are writing a program with a number of procedures, it is a good idea to include a description of the program. On the Flip side of the page, move the cursor to the top and type a description. You might type something like

> <<The following program draws a house with a "For Sale" sign. Type HOUSE.AND.SIGN to run the program.>>

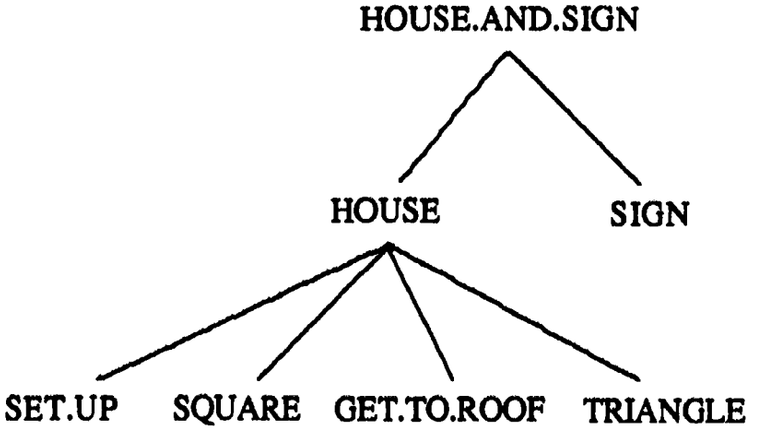
The "<<" and ">>" symbols are to set off the description or set of directions from the procedures themselves. You can put almost any text you want on the Flip side of the page. Simply be careful to follow two rules.

1. Be sure to put any text that is not part of procedures *after* any ENDs and *before* any TOs. Otherwise, Logo will think the text is part of a procedure.

2. Don't begin any line of text that is *not* part of a procedure with the word "to." The word "to" is reserved to tell Logo that a new procedure is contained in the text that follows.

A Procedure Tree

The procedure HOUSE.AND.SIGN is the *main procedure*, or *top-level procedure*. It uses a number of other procedures. These procedures are called *subprocedures* of HOUSE.AND.SIGN. A good way to visualize the relationships among procedures is by using a *procedure tree*. In a procedure tree, a subprocedure is drawn below the procedure that uses it. For example, the procedure tree for HOUSE.AND.SIGN looks like this:



Notice that the HOUSE procedure, which is a subprocedure of HOUSE.AND.SIGN, has four subprocedures of its own.

The word *superprocedure* is also commonly used to describe relationships among procedures. For example, HOUSE.AND.SIGN is a superprocedure of SIGN.

To summarize:
- HOUSE.AND.SIGN is the main, or top-level procedure.
- HOUSE and SIGN are subprocedures.
- HOUSE.AND.SIGN is a superprocedure of SIGN.
- HOUSE.AND.SIGN is a superprocedure of HOUSE.
- SETUP, SQUARE, GET.TO.ROOF, and TRIANGLE are subprocedures of HOUSE.
- HOUSE is a superprocedure of SETUP, SQUARE, GET.TO.ROOF, and TRIANGLE.

Notice how easy it is to see the relationships among procedures using a procedure tree. The written list of the relationships is harder to follow. The procedure tree gives you a visual image of the structure of your program. You should always draw a procedure tree when you are working with more than two or three procedures.

There are some "rules of thumb" that you should remember when you are drawing a procedure tree.

1. The name of the program (the procedure name you type to run the program) is at the top of the procedure tree on a line by itself.

2. Each subprocedure of the main procedure is listed on the next level (or line). These subprocedures are listed left to right as they occur in the main program. That is, the first subprocedure is listed on the left, the next subprocedure is listed to the right of the first, and so on.

3. A line connects each procedure with its subprocedure(s).

4. Each procedure in the program along with its subprocedures is represented using the rules given in step 2.

5. If a procedure is a subprocedure of more than one procedure, it should appear "under" each procedure from which it is called.

6. If a procedure "calls itself" (that is, includes itself, which is an idea not covered in this book but covered in more extensive Logo books), then the name of the procedure appears below the name of the procedure connected by a line just like any other subprocedure.

7. All "user defined" procedures (those that you write) should appear in the procedure tree.

8. No primitives (such as CG, FORWARD, or REPEAT) are listed in the procedure tree.

When you have completed your procedure tree, you should be able to tell exactly which procedures are called from any given procedure. You should be able to tell what procedures call any subprocedure. A carefully drawn procedure tree can be very helpful in solving many debugging problems you may have with your program.

Frequently Asked Questions

1. I don't see the sense in having a lot of subprocedures. Why not just write one big procedure?

   Answer: As you work on more complex problems, you will discover lots of answers to this question. One answer is that you can test a subprocedure by itself, and then fully debug it without dealing with the complexities of the whole program. A second is that use of subprocedures "modularizes" a problem–breaks it into chunks that your mind can deal with. A third answer is that it makes the overall program easier to read and to modify. Imagine writing an English composition without breaking it into paragraphs!

2. Is there some rule about how long a procedure should be?

   Answer: A procedure can be quite long–you are limited only by computer memory space. However, it is a mistake to write such long procedures because they are hard to read and hard to debug. As a rule of thumb, never write a procedure is longer than one page on the display screen.

3. Can I put more than one instruction on a line?

   Answer: Logo allows you to put many instructions on one line. However, writing more than one instruction on a line usually makes your procedures hard to read and debug. Resist the temptation to put several instructions on a line "just this once" because before you know it you will have procedures that are quite difficult to read and debug.

4. What names should I give the procedures I write?

   Answer: Give your procedures meaningful names that describe exactly what each procedure does. It is easy to find yourself adding more and more commands to a procedure. Soon you have a very long procedure that is hard to debug. Perhaps the name of the procedure no longer describes the function of the procedure. Take time to subdivide and rename procedures as you work. It will save a lot of time later when you are debugging.

5. My program doesn't work, and I am having trouble figuring out what it does. I get lost within all of the different subprocedures. Help!

   Answer: Draw a procedure tree. It can be very helpful in seeing the overall structure of the program. Then mentally, using paper and pencil as an aid, follow through the instructions of each procedure one line at a time. Remember that you can hand trace or play turtle to help debug your program. It is often quite helpful to do this hand tracing with a friend. Often you or your friend will easily spot bugs that you couldn't find by yourself.

Bugs and Debugging: Breaking a Problem Into Small Parts

The use of superprocedures and subprocedures adds still another level of complexity to programming. You now have the power to attack really complex problems, and you now have the power to create really complex bugs that are difficult to find. With little effort you can create a "mess" of intertwined procedures full of bugs and nearly impossible to debug.

It is easy to conceptualize a project whose details overwhelm your mind. If the house example is not complex enough, add a chimney, a door, some windows, a person's face in the window, some trees, some flowers, some clouds, and so on. The overall concept of a picture of a house is simple, but the number of details you might add to the program is very large.

The way to attack a complex problem is to break it into many small pieces. Write a subprocedure for each small piece, test each subprocedure to eliminate bugs, and then begin to put the pieces together. This requires a very systematic approach, quite a bit of paper-and-pencil work, and very careful attention to details. It also requires skill in breaking big problems into smaller problems.

For example, suppose that you want your house picture to include a number of trees and a number of flowers. You might imagine this as being accomplished by a LANDSCAPE procedure. The LANDSCAPE procedure contains a TREES procedure and a FLOWERS procedure. Will all of the trees be the same? If so, you will want a procedure that draws one tree, and the trees procedure will make repeated use of it. You should use a similar method for your flowers.

Each subprocedure should be of a size that you can easily hold in mind and that you can easily debug. A procedure tree shows the way the subprocedures fit together and thus indicates how to test groups of subprocedures. As you put the procedures together, start with the bottommost subprocedures and test them first. Gradually work your way up the procedure tree to the main procedure.

Computers and Problem Solving: Procedures to Solve Problems

This chapter illustrates several really important ideas in writing programs to solve problems:

1. Give a lot of thought to how you are going to attack a problem before you actually start to write a program. You may want to make use of pencil and paper as you conceptualize the problem and work toward a clear set of goals.

2. Learn to break a big problem into chunks of a size your mind can easily handle. Give the mind-sized chunks names that help you recall what each is designed to accomplish.

3. Test each subprocedure all by itself so that you are quite sure no subprocedure contains an error. Start with the procedures at the "bottom" of the procedure tree and work toward the top.

It is evident that each of these ideas is also quite important in solving problems without the use of a computer. George Polya (1957) made major contributions to the field of problem solving. He suggested a four-step approach that is useful in attacking almost any problem. Polya's four-step plan contains the following ideas.

1. Understand the problem. (What are you trying to accomplish? What is the goal? How can you tell if you have solved the problem?)

2. Develop a procedure you feel will solve the problem, or find a procedure through appropriate use of reference materials.

3. Carry out the steps in the procedure, paying careful attention to avoiding errors.

4. Carefully examine the final results. Has the original problem been solved? If not, remember that you may have used an incorrect procedure or you may have made a mistake in carrying out the steps of the procedure.

Activities

1. Now that you know how to write multiple procedures, plan a picture of your choice and write the procedures to create it. Move along step by step like the example in this chapter. Draw a

procedure tree as you go. Be sure to keep the individual procedures short. You should not have any procedures longer than 20 lines, with one instruction on each line. If you put too much into a procedure, it becomes hard to find errors. Here are some suggestions:

- Add to the HOUSE.AND.SIGN project begun in this chapter.
- Write a program to draw a greeting card with a border around the edge and a greeting in the middle.
- Write a program to place polygons on the screen filled with different shapes or colors.
- Write a program that places randomly colored "stars" all over the screen. Perhaps include a row of houses "below" the sky.
- Develop a set of procedures useful in designing the layout of a room.

As you work on your project, document your thinking in your journal. What programming style do you use? How does the approach you take to your project affect the final outcome? What did you learn as you worked on your project?

2. Write a program that displays "paced" text. That is, you might write

```
TO START.POEM
PRINT [The fog comes]
WAIT 10
INSERT [On little]
INSERT CHAR 32
WAIT 5
INSERT [cat]
INSERT CHAR 32
WAIT 5
PRINT [feet]
WAIT 10
END
```

Adjust the "timing" to suit your interpretation of the text you choose. Perhaps you'll even want to include some graphics.

3. Experiment with animation. That is, you can make the turtle appear to move by typing

```
PU
REPEAT 20 [FORWARD 2 WAIT 1]
```

Write a program that includes animation. Perhaps you'll want a background with a shape of your choice moving across it—maybe a helicopter can fly by or a kitten can run across the screen.

4. Polya developed the four-step approach to problem solving before people began to use computers. Examine the steps carefully. Can you think of a problem where these ideas do not apply? Are they applicable in writing computer programs? That is, does Polya's four-step plan contribute to transfer of learning between problem solving with and without computers?

Chapter 12
A Word About Designing Programs

There are a number of ways to go about creating programs. Different people prefer different approaches. Different computer languages lend themselves to different methods. Now that you are beginning to write programs containing a number of procedures, you should spend some time thinking about the ways to design a program. You should be examining your own programming to see which method(s) suit your particular learning style and problem solving style.

Top Down Programming

Top down design refers to writing a program "from the top." Think about the HOUSE.AND.SIGN program from the previous chapter. If you use a top down approach, the first procedure you write would be

```
TO HOUSE.AND.SIGN
HOUSE
SIGN
END
```

Next you would write the HOUSE procedure.

```
TO HOUSE
SET.UP
SQUARE
GET.TO.ROOF
TRIANGLE
END
```

Then you would write SET.UP, SQUARE, GET.TO.ROOF, and TRIANGLE. Finally, you would write SIGN.

With this method, you plan the whole and then write ever smaller parts. This is a very regimented and structured approach to problem solving. You must do a lot of planning in advance. You need to know exactly what you want to accomplish before you even begin. This approach allows little room for exploration as you go along.

It has another difficulty. You cannot see what each procedure does as you write it. If you have written just HOUSE.AND.SIGN and try to run it, you get the message

```
I don't know how to HOUSE in HOUSE.AND.SIGN
```

This tells you that the procedure HOUSE.AND.SIGN at least begins to work right, but it doesn't show you any meaningful results. However, top down programming is a very powerful approach to solving problems in a computer environment. In some programming languages, the top down approach is the easiest to use. As a novice Logo programmer, you will most likely find that you are more comfortable using a bottom up approach.

Bottom Up Programming

In bottom up programming, you start with small pieces and build them up into a complete program. If you used bottom up programming in the HOUSE.AND.SIGN program, you wouldn't start with a complete plan. Instead you might start with a general idea of what you wanted to accomplish and an idea of how to make some progress on some major pieces. You might first create the SQUARE procedure and then the TRIANGLE procedure. After some fiddling with these, you might discover that, when put together, they make a nice house. That might lead to the HOUSE procedure.

You might be experimenting with the PRINT command and get the idea to write a procedure to put a sign next to the house. Next would come some experimenting to create the SIGN procedure. Last of all, the HOUSE.AND.SIGN procedure would be written.

With this method, there is less advance planning. You play with an idea, turn it into a procedure, and let that idea lead you to another. You may have little or no idea where you are going as you begin. You are like an artist, being led along by your own creation and creativity. Logo lends itself to the bottom up style of programming better than most programming languages.

Much of what is written about Logo implies that the only "right" way to program in Logo is bottom up. This emphasis on bottom up style grows out of the philosophy of discovery learning that is closely allied with Logo. It is assumed that the Logo programmer will experiment with an idea and then use that idea as part of a larger project.

However, in the first part of this chapter, you saw how one might write a program using a top down approach to problem solving in Logo. Professional programmers usually use a combination of top down and bottom up techniques. Both are taught in a modern computer science department.

Middle Out Programming

Very few people actually program *either* in a true top down or bottom up style. Most people use some combination of methods for writing programs, which might be described as "middle out." Middle out is the method that was actually used for writing the HOUSE.AND.SIGN program. There was an initial idea of what the program should do. Then small parts were written. Finally all the pieces were put together. Some things were done top down, other things bottom up, but most things were done from the middle, working both towards the top and the bottom.

Polishing Your Program

The programming style you use to do a project should be adapted to your capabilities and the nature of the problem. However, it is best if the final program looks structured and well organized, much as if it had been done in a top down manner. That is, there should be a main procedure with a title that describes the entire program. This main procedure should be made up primarily of procedure calls whose titles describe the subparts of the program. Many of the subprocedures in the program may also be made up of procedure calls that describe even smaller parts of the program.

Both the programming process *and* the final product are important in any programming project. During the development of a project, you probably focus on writing the code and on debugging the results. However, when you have finished a program in Logo, it should be polished for "publication," just as you polish your final draft of a written report before you turn it in to a teacher or employer.

What is the "correct" style to use when writing a Logo procedure? Are there "rules" that you should follow? The issue of programming style is one for which there is very little agreement within the Logo community. At one end of the spectrum are those who feel that to impose *any* rules for writing procedures is inappropriate. They feel that such rules will interfere with creativity and exploration.

The authors of this book suggest an approach nearer the other end of the spectrum. For example, we suggest that an individual procedure should not exceed one screen in length and that you should not write more than one instruction on a line. It took professional programmers many years to discover that these are very useful guidelines. It is unreasonable to expect beginners to discover them for themselves.

Any Logo programs you write should be polished for publication, presentation, and preservation no matter who is going to see them. Polishing the final product makes it more readable both to yourself and to others. You will be surprised how quickly you can forget the details of a program you spent many hours writing. Only weeks later, a poorly written program can be very hard to untangle.

Here are some guidelines to help you begin developing a readable programming style.

1. Procedures should have meaningful names. That is, the name of the procedure should describe *exactly* what happens in that procedure.

2. There should be no more than one instruction per line.

3. No procedure should be longer than 20 lines (approximately the number of lines that fit on the page at one time).

4. The main or top level procedure should contain only user-defined procedure names. That is, you should not use a LogoWriter primitive in the main procedure. (This is an artificial rule that may be unnecessarily strong. Experience suggests that by keeping the numer of primitives in the main procedure small, good modular style is encouraged.)

5. If an instruction is longer than one line, you should indent the second and subsequent parts of the line in a logical and readable manner.

6. If you continue your work with Logo beyond the content of this book, you will learn about procedure inputs and variables. When you reach that stage of learning, it is important that you use meaningful names for inputs and variables that describe their function, just as you use meaningful names for procedures.

These rules are perhaps unnecessarily rigid for your long-term work with Logo programming. However, they are designed to give you specific guidelines to follow from the outset that will help you in debugging your future, more complex programs. As you continue your work with Logo beyond this book, you will no doubt relax or modify these rules in ways that are comfortable for you.

Programs written using the above guidelines are much easier to work with. Other people can easily tell what each part does. You can more easily make changes in your own work at a later date. Most importantly though, is that following these style guidelines contributes to ease in debugging. Sometimes the bugs are easy to find, sometimes they aren't. If the program is written well, then the debugging process takes less time.

Bugs and Debugging: Good Programming Style Helps

In the early history of computer programming the problems that were being attacked were rather simple programming tasks. The computers being used had relatively small memory capacity, and the programs that were written were modest in size and complexity.

Gradually the complexity of problems being attacked grew, as did the speed and memory capacity of computers. Teams of programmers began to work together to attack problems too big for a single programmer. This led to the problem of errors in one part of a program, written by one programmer, messing up results produced in a different part of the program being written by a different programmer. Needless to say, it became a programming and debugging nightmare.

Out of these circumstances a discipline of computer programming slowly began to emerge. The need for a careful top down analysis and very careful modularization of large tasks became apparent. Programs needed to be carefully structured if they were going to be easy to debug and easy to modify as the need arose. Programming languages were developed that supported such an approach to programming. Computer science departments began to teach structured programming.

As computers began to become common in precollege education, the issue arose of the needed qualifications to teach computer programming. It was evident that anyone who knew a little programming could teach programming. However, it also became evident that usually a person knowing only a little programming did a very poor job of teaching the key ideas of top down analysis, structured programming, and writing bug-free programs that were being taught in college computer science departments.

This short book has placed considerable emphasis on key ideas for preventing, detecting, and correcting bugs. At some future date you may find yourself helping students to learn to write Logo programs. We hope that you will place considerable emphasis on helping your students learn about bugs and debugging as well as using good programming techniques and problem solving strategies.

Computers and Problem Solving: Programming and Problem Solving Strategies

A *strategy* is a plan of action that might be helpful in attacking a certain type of problem. This chapter mentions three programming strategies: top down, bottom up, and middle out. The previous chapter mentioned Polya's four-step strategy for attacking almost any problem.

There are many strategies that are useful in writing computer programs to solve problems. For example, the strategy of testing procedures, working from the bottom up, is often quite useful in debugging a main procedure. The strategy of giving procedures names that are meaningful in the context of the problem to be solved will often help you write a more easily readable program, and thus write a program that is easier to debug.

Once you start thinking about it, you will discover that you know lots of strategies that you routinely use. Here is an example. Is there some particular day of the week when you do laundry? Is there some particular filling station where you usually buy gas for your car? Think of routine decisions you automatically make, often with little conscious thought. In each case you make use of a strategy of "do what worked before."

What is your strategy for studying for a test? What is your strategy for getting a term project done on time? What is your strategy for making new friends? What is your strategy for dealing with hard problems—for example, with when you are stuck on a Logo problem?

Research on problem solving strongly supports the idea that students should learn about strategies. They should learn to recognize the types of strategies they are using. They should gain skill in recognizing when a particular strategy is effective and when it isn't. These ideas can all be taught in a Logo programming environment. However, the goal is to help students transfer such knowledge of strategies to other fields. This requires the examination of lots of different strategies and their possible use in a variety of fields. It requires having students think about, talk about, write about, and practice using a variety of strategies.

The research literature on problem solving strongly supports the idea of helping students understand the strategies they use and helping them to gain more strategies. However, the literature is not so clear on the best approach to use. Some educators feel that the best approach is a discovery learning approach. Students should be put in environments in which they will discover useful strategies. Then the teacher can help make these discoveries more explicit. A substantially different approach is to give students explicit direct instruction on the strategies you want them to learn. Most leaders in the Logo community advocate the former approach.

Activities

1. Select a project that involves writing a number of procedures. As you are working on it, examine your own programming style. Which of the styles mentioned above best describes the way you prefer writing programs? Be sure your final program is polished for publication.

2. It is sometimes suggested that artists usually use a bottom up strategy in solving problems while mathematicians use a top down approach. The ideas of inductive and deductive logic seem related to this. What do you feel about his? Write about it in your journal.

3. Spend some time jotting down problem solving strategies in your journal. Then share your list with other members of the class. Did your classmates think of strategies you did not have on your list? Do other people use strategies you never use? What strategies seem most universal among people in your class? Are there some strategies that are more important to teach to students than others?

4. Notice that many of the programming assignments in this book are quite open ended. In essence, they ask you to create a problem and then solve it. How does this instructional strategy compare with instructional strategies you have encountered in other courses? Do some journal writing on advantages and disadvantages of each of these instructional strategies. Be aware that "problem posing" is considered to be a very important part of the field of problem solving.

Chapter 13
Music

The preceding chapters introduced you to the rudiments of LogoWriter. There are many features of LogoWriter we have not discussed. There is also much more to be learned about programming. Should you choose to continue your work with Logo, you will no doubt explore many of these. However, we are going to conclude this book with a section on music. Writing a program to create music is an excellent way to practice some of the new programming techniques you have learned in the last two chapters.

Try typing

```
TONE 440 50
```

You hear a musical note. Try changing the numbers after TONE. What does each one do?

Did you discover that the first number is the pitch of the note? The bigger the number, the higher the note. The second number is the length or duration of the note. The bigger the number, the longer the note.

The duration of a note is measured in twentieths of a second. That is,

```
TONE 440 20  plays for 1 second
TONE 440 60  plays for 3 seconds
TONE 440 10  plays for 1/2 second
```

To compute the length of a note, multiply the second input to TONE by 1/20.

The frequencies of the notes are the actual frequencies given in vibrations per second. The table below gives the values for a number of octaves:

OCTAVES	NOTES											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
1			37	39	41	44	46	49	52	55	58	62
2	65	69	73	78	82	87	92	98	104	110	117	123
3	131	139	147	156	165	175	185	196	208	220	233	247
4	*262	277	294	311	330	349	370	392	415	440	466	494
5	523	554	587	622	659	698	740	784	830	881	932	988
6	1047	1109	1176	1244	1319	1398	1480	1566	1663	1761	1864	1973
7	2095	2213	2346	2495	2637	2797	2959	3142	3327	3510	3743	3946

*This is Middle C

Using the above chart, you would play a C scale (the white notes on the piano) for the octave starting with middle C using these commands:

```
TONE 262 10
TONE 294 10
TONE 330 10
TONE 349 10
TONE 392 10
TONE 440 10
TONE 494 10
TONE 523 10
```

There are no flats in this scale. If you have music that uses flats, you need to know, for example, that A sharp is the same note as B flat. (Black notes on a piano are used to play sharps and flats.) You then use the chart accordingly.

Suppose you want to write a program to play a simple tune. You can write a procedure for each line of the song. For example, to have LogoWriter play "Three Blind Mice," you can begin with

```
TO THREE.BLIND
TONE 330 10
TONE 294 10
TONE 262 20
END
```

This phrase is repeated twice, so you can write

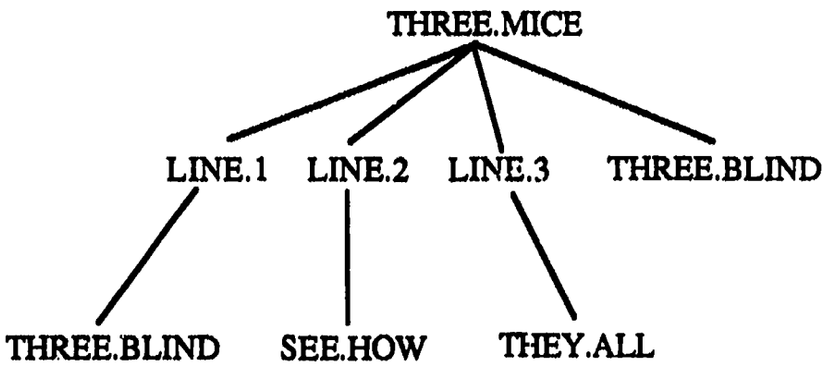
```
TO LINE1
REPEAT 2 [THREE.BLIND]
END
```

Similarly, the second line can be written as follows:

```
TO LINE2
REPEAT 2 [SEE.HOW]
END

TO SEE.HOW
TONE 392 10
TONE 349 5
TONE 349 5
TONE 330 20
END
```

Can you finish this song? The procedure tree might look like this:



Using some words from the song helps you keep track of what procedures play what part of the song. In each of the "LINE" procedures above, a phrase of music is repeated, sometimes with the same words, sometimes with different words.

Even if you are not "musical" you can write music using LogoWriter. With some simple sheet music and a chart to translate the symbols on a musical staff into frequencies, you too can be a music performer and composer!

Some Sound Effects

The TONE command can be used for sound effects as well as music. Experiment with different numbers for the frequency. Try small numbers to get buzzes

```
TONE 40 10
```

and large numbers to get squeaks

```
TONE 9000 20
```

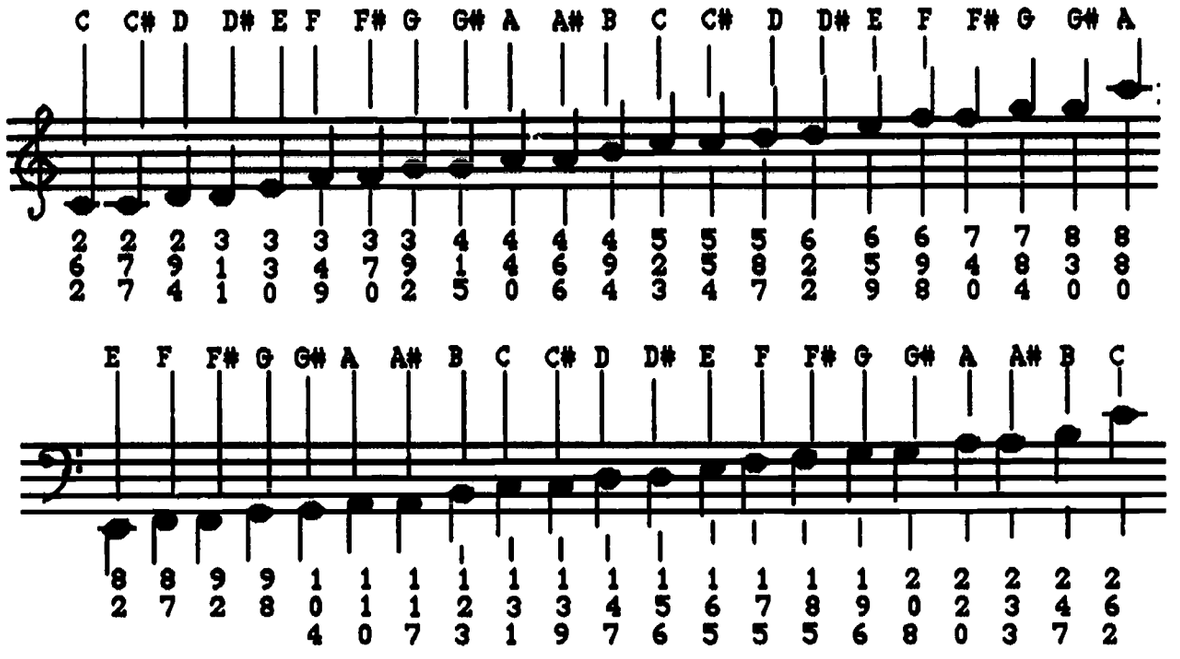
and slowly changing numbers for an interesting effect

```
TONE 100 1
TONE 105 1
TONE 110 1
TONE 120 1
TONE 125 1
```

The sounds you can make using the TONE command can be quite varied.

Are You "Musical?"

Many people say "Oh, I'm not musical. I can't use Logo to make music." In fact, it is much easier to use a computer to make music than it is to play the piano or most instruments. To get started, find some beginning instrumental music. You might ask the music department in your school for some beginning music in the key of C. Match the notes on the music with the following chart.



When you have identified the notes by name (A, B, C, etc.), use the chart to write the frequencies next to each note. You will want to experiment with the length of a "quarter note," but the following chart shows you the relationship among notes.

Note	Name	Duration
♪	Eighth note	8
♩	Quarter note	16
♩.	Dotted quarter note	24
𝅗𝅥	Half note	32
𝅗𝅥.	Dotted half note	48

Then you are ready to enter any song for which you have music.

Frequently Asked Questions

1. When I type the TONE command, I hear nothing even though I tried a number of different inputs for the frequency and duration. What's wrong?

   Answer: Check the computer system you have. Does it have a volume control? Be sure that it is not turned off. Does it have an earphone jack? Unplug (or use) any earphones plugged in to the computer. Does the computer "beep" when you start LogoWriter? If not, perhaps the speaker in your computer is broken or disconnected.

2. I want to play more than one note at a time. How can I do that?

   Answer: Even if your computer system has "multiple voices," current versions of Logo only allow you to play one note at a time. Sometimes people try putting the melody of a song on one computer and the harmony on another and try to run them at the same time. Generally this does not work well because of the slight differences of speed of two different computer systems and possible differences of the information stored in the memory of Logo.

3. I want to play music and have a shape move at the same time. How do I accomplish this?

   Answer: The answer to this question is not simple. You can't simply write a MOVE procedure and a PLAY procedure because if you type

```
MOVE
PLAY
```

   your shape will move, and after it has stopped your music will play. You must instead, alternate notes and movement. For example.

```
TO PLAY.AND.MOVE
TONE 330 10
FORWARD 5
TONE 294 10
FORWARD 5
TONE 262 10
FORWARD 5
END
```

   While this approach works fairly well, it is particularly easy to end up writing long programs that are difficult to debug. Think carefully about what you want to put in each procedure.

4. The music I create using LogoWriter sounds off pitch. What am I doing wrong?

   Answer: Many computers on which versions of Logo run were not designed to produce sophisticated sound. Generally, Logo does not access the more sophisticated sound capabilities available on newer computer systems. Thus, it is not unusual for Logo music to sound off-pitch in the upper and lower pitch ranges. It is generally best to use the middle range of notes to create music.

Bugs and Debugging: Working With Sound

Debugging programs involving sound require a different set of skills than you have previously used with Logo. You must listen to a procedure to be able to tell if it is working correctly. Often nothing appears on the screen. There is no graphics and there is no text. You must hold the sounds in your mind and then adjust your program accordingly. It is particularly important that you design your musical programs modularly. If each individual procedure is short, you can debug each one separately much more easily.

Music is a natural environment for making use of modularity. Music itself is divided into notes, measure, lines, and pages. Use these natural subdivisions to plan your procedures. If your music has words, making use of the words for procedure names can often speed the debugging.

Computers and Problem Solving: Music and Domain Specificity

At several previous spots in this book we have talked about the idea of domain specificity in problem solving. We have suggested that the computer is a tool useful in helping to solve problems in many different domains.

The music examples in this chapter allow you to explore domain specificity in relation to using a very general-purpose tool as you attempt to solve some problems in music. Suppose, for example, that you did not know how to play any musical instrument or know any musical notation. Would it still be possible for you to compose some music? How could a computer help?

It turns out that you know quite a bit about music even if you don't play any musical instruments and have never composed any music. This is because you have been exposed to music throughout your life. Your brain has processed a great deal of music and "knows" what sounds right.

The computer can be used both as an aid to composing and as an aid to performing music. The composing can be done in the simple Logo programming notation illustrated in this chapter. The performance is done automatically by the computer. With the Logo NOTE command and the computer performance capabilities, you can compose and perform music.

You should realize, of course, that there are much more sophisticated computer musical notation systems and much better computer music performance systems. It is now common for professional composers to make use of these aids to composition. However, even with an inexpensive microcomputer and Logo PLUS it is possible to introduce young children to composing with the aid of a computer. This certainly brings a new dimension to the elementary school music curriculum.

Computers and Problem Solving: Society, Education, and the Future

The invention of reading and writing profoundly changed the societies of our world. Reading and writing are tools of the human mind. They expand its capabilities to solve complex problems. They make it easier to share intellectual progress–to preserve and pass on knowledge from one generation to the next. They contribute greatly to building on the previous work of other people as well as on one's own previous work.

It is interesting to compare use of pencil, paper, and books with the computer. They share many characteristics. For example, they are all aids to the human mind and provide ways to build on previous work of others. Each has certain advantages and certain disadvantages. For example, pencil, paper, and books are cheaper and more portable than a computer. However, writing done on a computer is more easily edited than writing done using paper and pencil. In addition, a computer can make use of an automatic spelling checker. (And you should keep in mind that pencil, paper, and books do not contain an analogue to the animated graphics and sound capabilities of computers. Clearly, computers bring a new dimension to the storage, processing, and retrieval of information.)

From your earliest childhood you have been exposed to books, pencil, and paper. Much of your education has focused on learning to read, write, and solve problems in this environment. You have been functioning in this environment for so long that you probably are not even consciously aware of how you use these facilities to help you solve problems.

As you develop your computer skills, you are working both on learning specific details of using a computer and the harder problem of learning to think and solve problems in a computer

environment. It may take you years of using a computer before you become as comfortable with it as you are with pencil and paper.

Computers are gradually becoming commonplace, even in elementary schools. Many children are growing up in household environments where they begin to use computers simultaneously with or even before they begin to use pencil and paper. It seems clear that such children will have a different view of computers than people who are first introduced to this tool as adults.

Reading and writing were invented many thousands of years ago. It took many thousands of years for people to develop the technology to mass produce books cheaply. It took thousands of years for educational systems to develop that could adequately address the issue of helping all children to become literate. Even now we do not have a 100 percent success rate. Many students experience considerable difficulty in learning to read and write.

From an historical perspective, we are at the very beginning of the development of computer use. The capabilities of computers are continuing to grow quite rapidly. Our knowledge of how to help children learn to make effective use of computers is still quite limited. Very few children have grown up in computer-rich environments, surrounded by adults who use computers freely and easily.

It seems clear that computers are changing our society. Their impact on our educational system is just barely beginning to be felt. The impact will be cumulative as we learn more about computers in education, as computer facilities become better and more readily available, and as more children grow up in a computer-rich environment. This will be a continuing challenge to current educators!

This book has provided you with a very brief introduction to one particular computer programming language. However, many of the ideas that have been presented are independent of any particular programming language or computer hardware. Thus, you have a solid foundation on which to build your future computer knowledge. The authors of this book wish you the best of luck in your future computer studies and your use of computers.

Activities

1. If you are involved with music outside of this class, pick a piece of music that you have sung or played and "teach" LogoWriter to play it!

2. Writing music in LogoWriter is an excellent way to practice writing modular programs. Select a piece of music and spend some time dividing either the written musical score or the words into small, meaningful sections. Each section will then be a procedure. Draw a procedure tree for your song. Then, when you write your song you will have an easy-to-debug program because you can check each part of the song as you go. (Note that this is using a strict top down approach to solving the problem of writing your song in LogoWriter.) Keep track of your planning style in your journal.

3. Try adding music to a page you have previously created. For example, you might have a birthday greeting card that plays "Happy Birthday."

4. Try writing a program that plays the music of a song as it displays the words. Hint: You'll need to use PRINT, INSERT, and WAIT interspersed with your TONE commands. Be careful to subdivide your program in a meaningful way.

5. Experiment with "sound effects" using TONE. See if you can get some interesting sounds that you can then put into a larger program. Note that your exploration with TONE is likely to be

bottom up. Be sure to keep some notes in your journal. How can you describe the sounds you make by just using words?

6. Do some journal writing on similarities and differences between the book and the computer. Brainstorm on how computers might eventually contribute to major changes in our educational system.

7. Some people argue that all children should learn something about computer programming, while others argue that computer applications software, such as word processors, spreadsheets, and databases, have obviated the need to learn about computer programming. What is your personal opinion? Discuss this in your journal.

APPENDIX

DESCRIPTION OF LOGOWRITER KEYS

	APPLE	IBM	COMMODORE
General Purpose Keys			
Flip	Apple-F	Ctrl-F	Commodore-F
Flips a page over			
Up	Apple-U	Ctrl-U	Commodore-U
Moves the cursor up from Command Center			
Down	Apple-D	Ctrl-D	Commodore-D
Moves cursor down into the Command Center			
Esc	Esc	Esc	Esc
Used to leave a page or a special mode			
Stop	Apple-S	Ctrl-Break	Commodore-S
Stops program or instruction; returns to Command Center (See also Esc.)			
Help	Apple-0	Fn-10	Commodore-0
Gets the Help page			
Contents Page Keys			
Up and Down Arrows	Up & Down Arrows	Up & Down Arrows	Up & Down Arrows
Moves cursor up and down through page names that are in the scrapbook			
Top of Page	Apple-up arrow	Home	Commodore up arrow
Moves the cursor to the top of the Contents list			
Bottom of Page	Apple-down arrow	End	Commodore down arrow
Moves the cursor to the last page name in the Contents list			
Return Enter	Return	Enter	Return
Chooses the page that the cursor is on.			
Erase to end of Line	Apple-6	Fn-6	Commodore-6
Erases the page named on the line that the cursor is on -- permanently!			
Graphics Keys			
Turtle-Move	Apple-9	Fn-9	Commodore-9
Makes it possible to move the turtle with the arrow keys; Esc to leave			
Label	Apple-8	Fn-8	Commodore-8
Text typed is added to the picture; use arrow keys to move; Esc to leave			

Word Processing Keys

Select	Apple-1	Fn-1	Commodore-1
Starts select mode; use arrows to select			
Cut	Apple-2	Fn-2	Commodore-2
Removes selected text; puts on the clipboard			
Copy	Apple-3	Fn-3	Commodore-3
Puts copy of selected text on the clipboard			
Paste	Apple-4	Fn-4	Commodore-4
Puts contents of the clipboard at the cursor position			
Erase to End of Line	Apple-6	Fn-6	Commodore-6
Erases all text cursor from cursor to the end of line			
Next Screen	Apple ->	Pg-Dn	Commodore->
Displays the next screen full of text			
Previous Screen	Apple <-	Pg-Up	Commodore<-
Displays the previous screen of text			
Top of Page	Apple-Up arrow	Home	Commodore-up arrow
Moves the cursor to the beginning of the text on the page			
Bottom of Page	Apple-down arrow	End	Commodore-down arrow
Moves cursor to the end of the text on the page			
Beginning of Line	Apple-B	Ctrl <-	Commodore-B
Moves the cursor to beginning of the line the cursor is on			
End of Line	Apple-E	Ctrl ->	Commodore-E
Moves the cursor to the end of the line; to where the next Return Enter was typed			
Open a line	Apple-O	Insert	Commodore-O
Opens new line after current cursor position			
Delete to the right	Control-D		
Deletes the character under the cursor			

Shapes Page Keys

Esc	Esc	Esc	Esc
Returns you to where you were when Shapes was chosen			
Flip	Apple-F	Ctrl-F	Commodore-F
Switches between the front (all shapes) and individual shapes			

On the flip side of the page only:

Next Screen	Apple ->	Pg-Dn	Commodore->
Displays the next shape for editing			
Previous Screen	Apple <-	Pg-Up	Commodore<-
Displays the previous shape for editing			
Arrow keys	Arrow keys	Arrow keys	Arrow keys
Moves the shape cursor around the shape			
Space Bar	Space bar	Space bar	Space bar
Empties or fills a space within a shape			
Cut	Apple-2	Fn-2	Commodore-2
Clears a shape and stores it in memory			
Copy	Apple-3	Fn-3	Commodore-3
Stores a copy in memory without erasing it.			
Paste	Apple-4	Fn-4	Commodore-4
Puts a shape that been placed in memory into the shape that is displayed			
Stop	Apple-S	Ctrl-Break	Commodore-S
Cancels editing; restores to shape there before editing began			

SUMMARY OF KEYS: APPLE IIe/IIc

General Purpose Keys

Flip	Apple-F
Up	Apple-U
Down	Apple-D
Quit, leave page	Esc
Stop	Apple-S
Help	Apple-0

Word Processing Keys

Select	Apple-1
Cut	Apple-2
Copy	Apple-3
Paste	Apple-4
Erase to End of Line	Apple-6
Next Screen	Apple -->
Previous Screen	Apple <--
Top of Page	Apple-Up Arrow
Bottom of Page	Apple-Down Arrow
Beginning of Line	Apple-B
End of Line	Apple-E
Open a line	Apple-O
Delete right	Control-D

Shapes Page Keys

Quit, leave page	Esc
Flip	Apple-F

On the flip side of the page only:

Next Screen (Next Shape)	Apple -->
Previous Screen (Previous Shape)	Apple <--
Up, Down, Left, Right	Arrow keys
Cut	Apple-2
Copy	Apple-3
Paste	Apple-4
Make or Erase Block	Space Bar

Graphics Keys

Turtle-Move	Apple-9
Label	Apple-8
Leave Label or Turtle-Move	Esc

Contents Keys

Top of Page	Apple-Up Arrow
Bottom of Page	Apple-Down Arrow
Remove a Page (Erase to end of line)	Apple-6

SUMMARY OF KEYS: IBM

General Purpose Keys	
Flip	Control-F
Up	Control-U
Down	Control-D
Quit, leave page	Esc
Stop	Control-Break
Help	Fn-0

Word Processing Keys	
Select	Fn-1
Cut	Fn-2
Copy	Fn-3
Paste	Fn-4
Erase to End of Line	Fn-6
Next Screen	PgDn
Previous Screen	PgUp
Top of Page	Home
Bottom of Page	End
Beginning of Line	Control <-
End of Line	Control ->
Open a line	Insert

Shapes Page Keys	
Quit, leave page	Esc
Flip	Control-F
On the flip side of the page only:	
Next Screen (Next Shape)	PgDn
Previous Screen (Previous Shape)	PgUp
Up, Down, Left, Right	Arrow keys
Cut	Fn-2
Copy	Fn-3
Paste	Fn-4
Make or Erase Block	Space Bar

Graphics Keys	
Turtle-Move	Fn-9
Label	Fn-8
Leave Label or Turtle-Move	Esc

Contents Keys	
Top of Page	Home
Bottom of Page	End
Remove a Page (Erase to end of line)	Fn-6

SUMMARY OF KEYS: COMMODORE

General Purpose Keys

Flip	Commodore-F
Up	Commodore-U
Down	Commodore-D
Quit, leave page	Esc
Stop	Commodore-S Help
	Commodore-0

Word Processing Keys

Select	Commodore-1
Cut	Commodore-2
Copy	Commodore-3
Paste	Commodore-4
Erase to End of Line	Commodore-6
Next Screen	Commodore -->
Previous Screen	Commodore <--
Top of Page	Commodore-Up Arrow
Bottom of Page	Commodore-Down Arrow
Beginning of Line	Commodore-B
End of Line	Commodore-E

Shapes Page Keys

Quit, leave page	Esc
Flip	Commodore-F

On the flip side of the page only:

Next Screen (Next Shape)	Commodore -->
Previous Screen (Previous Shape)	Commodore <--
Up, Down, Left, Right	Arrow keys
Cut	Commodore-2
Copy	Commodore-3
Paste	Commodore-4
Make or Erase Block	Space Bar

Graphics Keys

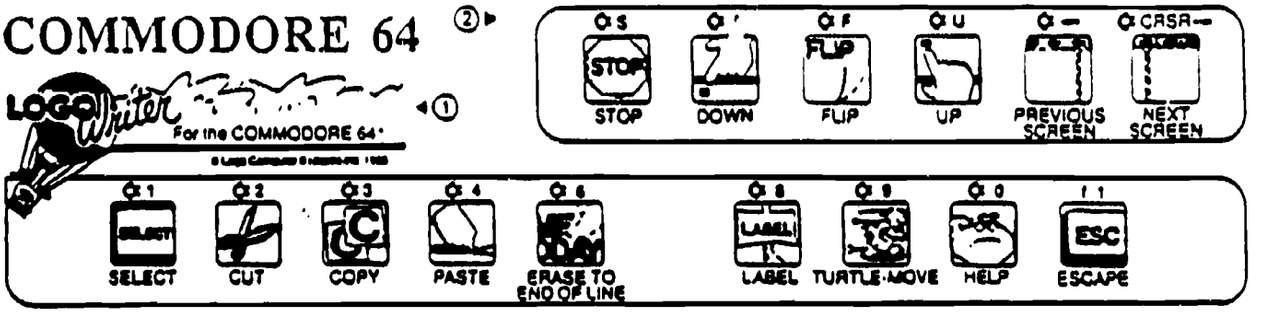
Turtle-Move	Commodore-9
Label	Commodore-8
Leave Label or Turtle-Move	Esc

Contents Keys

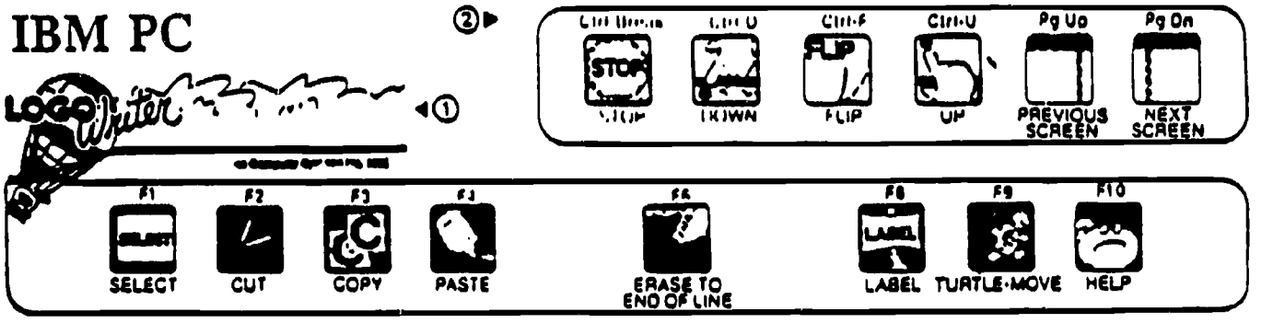
Top of Page	Commodore-Up Arrow
Bottom of Page	Commodore-Down Arrow
Remove a Page (Erase to end of line)	Commodore-6

KEYBOARD STICKERS

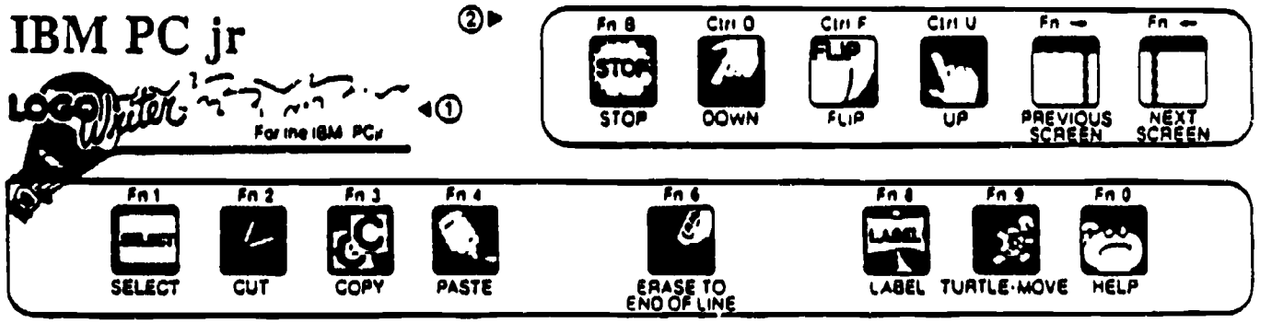
COMMODORE 64



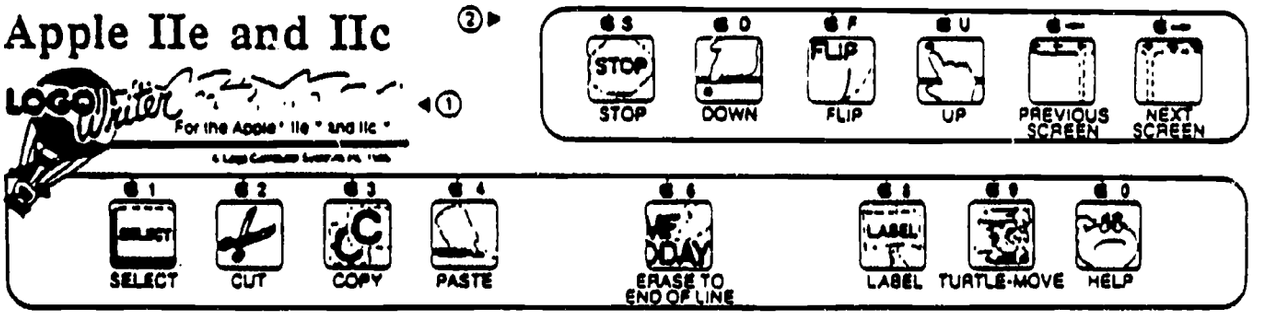
IBM PC



IBM PC jr



Apple IIe and IIc



These keyboard stickers are available for Logo Computer Systems, Inc. Call the sales office at 1-800-321-5646 for information on how to order.

SHAPES PAGES

1 2 3 4 5 6 7 8 9 10

11 12 13 14 15 16 17 18 19 20

21 22 23 24 25 26 27 28 29 30

INTERMEDIATE

1 2 3 4 5 6 7 8 9 10

11 12 13 14 15 16 17 18 19 20

21 22 23 24 25 26 27 28 29 30

PRIMARY

QUICK REFERENCE

LogoWriter Primitives and Special Words

Screen
CC - Clear Command center
CG - Clear Graphics
CP - Clear Page
CT - Clear Text

Scrapbook
CLEARTOOLS
CONTENTS
DOS (IBM only)
ERPAGE *pagename* - ERase Page
FLIP
* FRONT?
GETPAGE (GP) *pagename*
GETSHAPES
GETTOOLS *pagename*
LAST PAGE
LEAVEPAGE
LOAD *pagename*
LOCK
NAMEPAGE (NP) *pagename*
NEWPAGE
* PAGELIST
RESTORE
SAVEPAGE
SHAPES
* TOOLLIST
UNDO
UNLOCK

Workspace
RECYCLE
* SPACE

Pausing
WAIT *number*

Sound
TONE *frequency time*

Input
* BUTTON? *button number*
* KEY?
PADDLE *number*
* READCHAR
* READLIST (RL)
* READLISTCC (RLCC)

Assigning
CLEARNAME *word/list*
CLEARNAMES
MAKE *name word/list*
NAME *word/list name*
* NAME? *word*
PRINTNAMES
SHOWNAMES
THING *names*

Disk
* DISK
SETDISK *letter*

Events
CLEAREVENTS
WHEN *letter list to run*

Printer Commands
DSPACE – Double SPACE
PRINTSCREEN
PRINTTEST
PRINTTEXT80
SSPACE – Sincle SPACE

Special Words
END
FALSE
STARTUP
TO
TRUE

Math (Note: all are reporters)
\+ -
* /
> <
=
ARCTAN *number*
COS *degrees*
INT *number*
RANDOM *number*
ROUND *number*
SIN *degrees*
SQRT *number*

Flow of Control/Logic

* AND *true/false1 true/false2*
* IF *true/false list to run*
\# IFELSE *true/false list.to.run 1 list.to.run2*
* NOT *true/false*
* OR *true/false1 true/false2*
OUTPUT (OP) *word/list*
REPEAT *number list.to.run*
RUN *list.to.run*
STOP
STOPALL

Graphics

* ALL
\# ASK *turtle/turtle.list list.to.run*
BACK (BK) *number*
* BG
CHANGECOLOR *numbers*
list of three numbers (GS version)
* CHARUNDER - CHARacter UNDER
CLEAN
* COLOR
* COLORUNDER
* COLORVALUE *number* (GS version)
* CURSORPOS
* DISTANCE *list of two numbers* (GS/IBM)
EACH *list.to.run*
FILL
FORWARD (FD) *number*
HEADING
HOME
HT - Hide Turtle
LABEL *word/list*
LEFT (LT) *number*
PD - Pen Down
PE - Pen Erase
* POS
PU - Pen Up
PX - Pen Reverse
RESETCOLORS (GS version)
RG - Reset Graphics
RIGHT (RT) *number*
SETBG *number*
SETC *number*
SETH *number*
SETPOS *[x y]*
SETSH *number*
SETX *number*
SETY *number*
SHADE
* SHAPE
SLOWTURTLE (GS version)
ST - Show Turtle
STAMP
TELL *turtle/turtle.list*
* TOWARDS *list**
* WHO
XCOR
YCOR

Text Editing/Words and Lists

* ASCII *char*
BOTTOM
* BUTFIRST (BF) *word/list*
* BUTLAST (BL) *word/list*
CB - Cursor Back
CD - Cursor Down
CF - Cursor Forward
* CHAR
* CLIPBOARD
COPY
* COUNT
CU - Cursor Up
CUT
DELETE
* EMPTY? *word/list*
EOL
* EQUAL? *word/list1 word/list2*
* FIRST *word/list*
* FOUND?
* FPUT *word/list list*
* IDENTICAL? *word/list1 word/list2*
INSERT *word/list*
* ITEM *number word/list*
* LAST *word/list*
* LIST *word/list1 word/list2*
* LIST? *word/list*
* LPUT *word/list list*
* MEMBER? *word/list1 word/list2*
NEXTSCREEN
* NUMBER?
PARSE *word*
PASTE
PRESCREEN
PRINT (PR) *word/list*
REPLACE *word1 word2*
SEARCH *word*
SELECT
* SELECTED
* SENTENCE (SE) *word/list1 word/list2*
SETEXTPOS *number*
SETTC *number* (GS version)
SHOW *word/list*
SOL - Start Of Line
TAB
* TC (GS version)
* TEXTLEN
* TEXTPOS
TOP
TYPE *word/list*
UNSELECT
* WORD *word1 word2*
* WORD? *word/list*

ProDos Primitives

BYE
COPYFILE *name/pathname1 name/pathname2*
CREATEDIR *pathname*
* DIRECTORIES
ERASEFILE *name/pathname*
* FILELIST
LOADPIC *name/pathname*
LOADTEXT *name/pathname*
* ONLINE
* PREFIX
RENAME *name/pathname1 name/pathname2*
SAVEPIC *name/pathname*
SAVETEXT
SETPREFIX *pathname*
SETSLOT
* SLOT
* .VERSION

System Modifying Primitives

.BLOAD *name/pathname address*
.BSAVE *name/pathname address length*
.CALL *address*
.DEPOSIT *address byte*
* .EXAMINE *address*

IBM DOS Primitives

CHDIR *name/pathname*
COPYFILE *name/pathname1 name/pathname2*
* CURRENTDIR
* DIRECTORIES
DOS
ERASEFILE *name/pathname*
* FILELIST
LOADPIC *name/pathname*
LOADTEXT *name/pathname*
MKDIR *pathname*
RENAME*pathname name/pathname1 name/pathname2*
RMDIR
SAVEPIC *name/pathname*
SAVETEXT
* .VERSION

* Primitive is a reporter (Operation).
# Primitive is either a command or a reporter.

REFERENCES

Abelson, Harold and diSessa, Andrea. (1980). *Turtle geometry*. MIT Press: Cambridge, Massachusetts.

Beyer, B. K. (1983). "Common sense about teaching thinking skills." *Educational Leadership, 41*, (November), pp. 44-49.

Beyer, B. K. (April 1984). "Improving thinking skills: Practical approaches." *Phi Delta Kappan*, 556-560.

Beyer, B. K. (March 1984). "Improving thinking skills: Defining the problem." *Phi Delta Kappan*, 486-490.

Birch, Alison. (1986). *The Logo project book: Exploring words and lists*. Terrapin, Inc., Cambridge, Massachusetts.

Clayson, James. (1988). *Visual modeling with Logo: A structured approach to seeing*. MIT Press: Cambridge, Massachusetts.

Cory, Sheila and Walker, Margie et. al. (1985). *LogoWorks: Lessons in Logo*. Terrapin, Inc., Cambridge, Massachusetts.

de Bono, E. (1973). *Lateral thinking: Creativity step by step:* Harper Colophan Books, Harper and Row: New York.

ERIC. (December 1984). "Improving students' thinking skills." *The best of ERIC:* ERIC Clearinghouse on Educational Management, University of Oregon: Eugene, Oregon.

Fredericksen, N. (1984). "Implications of cognitive theory for instruction in problem solving." *Review of educational research, 54* , 363-407.

Gardner, H. (1983). *Frames of mind: The theory of multiple intelligences:* Basic Books: New York.

Gardner, Howard and Hatch, Thomas. (November 1989). "Multiple intelligences go to school: Educational implications of the theory of multiple intelligences." *Educational researcher*, 18, Number 8, 4-10.

Glatthorn, A. A. and Baron, J. J. (1985). "The good thinker." In Arthur Costa, ed., *Developing minds: A resource book for teaching thinking. 49-53*. ASCD.

Goldenberg, E. Paul and Feurzeig, Wallace. (1987). *Exploring language with logo*. MIT Press: Cambridge, Massachusetts.

Harvey, Brian. (1984). *Computer science Logo style: Volume 1: Intermediate programming* MIT Press: Cambridge, Massachusetts.

Harvey, Brian. (1986). *Computer science Logo style: Volume 2: Projects, styles, and techniques*. MIT Press: Cambridge, Massachusetts.

Harvey, Brian. (1987). *Computer science Logo style: Volume 3: Advanced topics*. MIT Press: Cambridge, Massachusetts.

Mayer, R.E. (1977). *Thinking and problem solving: An introduction to human cognition and learning*. Scott, Foresman, and Company.

Moursund, David G. (1990). *Getting smarter at solving problems*. International Society for Technology in Education: Eugene, Oregon. An extensive *Teacher's Manual* is also available.

Papert, Seymour. (1980). *Mindstorms: Children, computers and powerful ideas*. Basic Books, Inc.: New York.

Polya, G. (1957). *How to solve it: A new aspect of mathematical method*. Princeton University Press.

Rubinstein, M. F. (1986). *Tools for thinking and problem solving*. Prentice-Hall.

Specht, Jim. (1990). "Mathematics and writing–another look." The Writing Notebook.

Steinberg, E. R., Baskin, A. B. & Hofer, E. (1986). "Organizational/memory tools: A technique for improving problem solving skills." *Journal of educational computing research, 2* (2), pp. 169-87.

Sternberg, Robert. J. (1988). *The triarchic mind: A new theory of human intelligence:* Penguin Books.

Sternberg, Robert. J. (1990). "Thinking styles: keys to understanding student performance." *Phi delta kappan*, Volume 71 Number 5, pp. 366-371.

Torrance, E. P., & Torrance, J. P. (1973). *Is creativity teachable?* Phi Delta Kappa educational foundation: Bloomington, Indiana.

Turkle, Sherry. (1985). *The second self: Computers and the human spirit*. Simon and Shuster, Inc.: New York.

Watt, Daniel. (1984). *Learning with Apple Logo*. McGraw-Hill, Inc.: New York.

Watt, Molly & Watt, Daniel. (1986). *Teaching with Logo: Building blocks for learning*. Addison-Wesley Publishing Company: Menlo Park, CA.

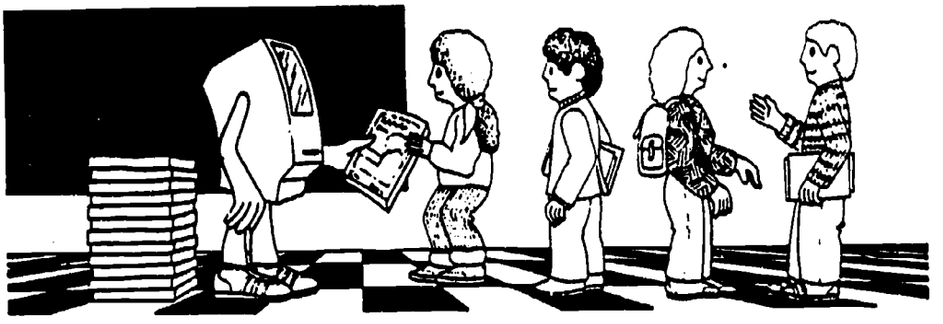
Weir, Sylvia. (1987). *Cultivating minds: A Logo casebook*. Harper & Row: New York.

Whimbey, A. (1984). "The key to higher-order thinking is precise processing." *Educational Leadership, 42*, (September), pp. 66-70.

Wickelgren, W.A. (1974). *How to solve problems: Elements of a theory of problems and problem solving*. W.H. Freeman and Company.

Yoder, Sharon K. (1990). *Introduction to programming in Logo using LogoWriter*. International Society for Technology in Education: Eugene, Oregon.

INDEX



Works users—seek help.

Practical and useful support is what you'll find in *Microsoft Works for the Macintosh: A Workbook for Educators* by Keith Wetzel. Step-by-step instructions thoroughly cover all the capabilities of MS *Works* including macros, spellchecking, and multiple column text and labels for Version 2.0. While you are learning, you'll create usable transparencies, letterhead, and other templates for use in your classroom or at home.

Microsoft Works for the Macintosh: A Workbook for Educators is available for both version 1.1 and 2.0. Please specify which version you need when ordering.

For pricing information or to order, contact: ISTE, University of Oregon, 1787 Agate St., Eugene, OR 97403-9905; ph. 503/346-4414.

Get help today. *Microsoft Works for the Macintosh: A Workbook for Educators* from ISTE.

We've polished up a proven favorite!

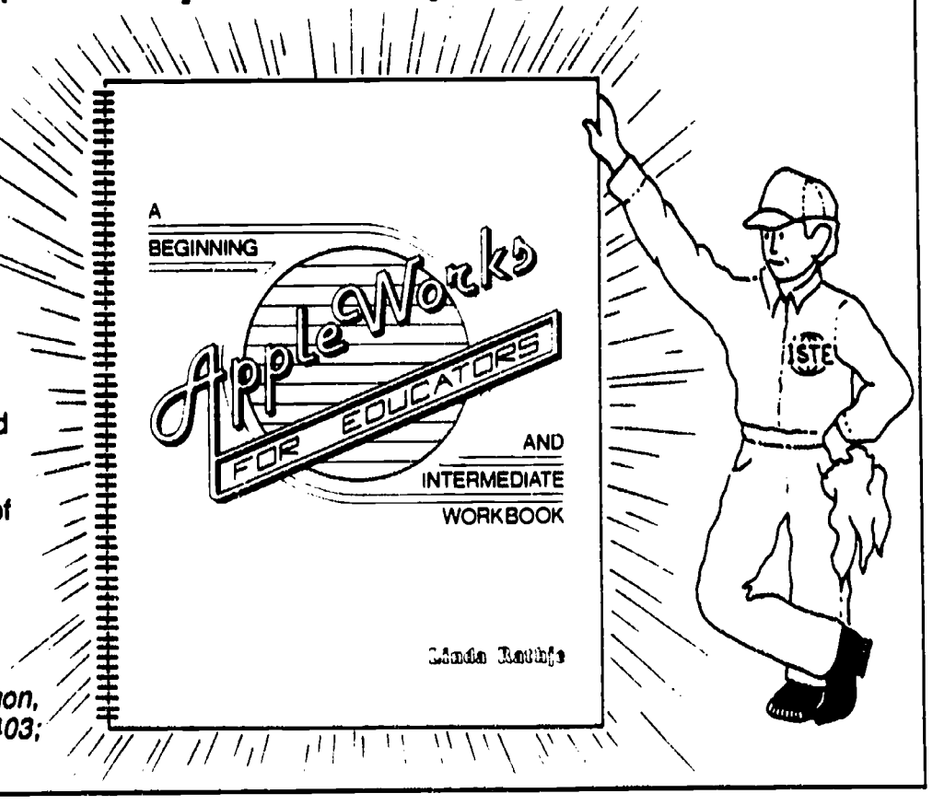
AppleWorks for Educators by Linda Rathje really shines. The new edition has been expanded to include sections for:

- mail merge
- integration activities
- a glossary, and
- software applications.

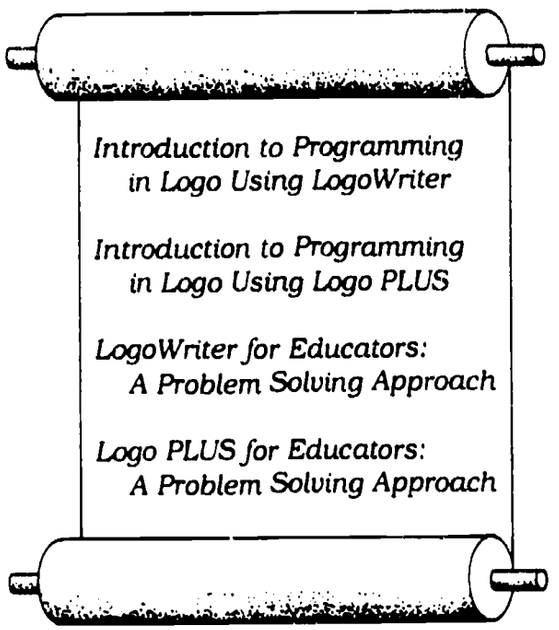
Each section provides step-by-step instructions. Beginning and intermediate AppleWorks® users learn word processing, database and spreadsheet management, and printer options.

Your copy includes a data disk of working examples. Add *AppleWorks for Educators* to your classroom and watch your students shine.

Contact: ISTE, University of Oregon, 1787 Agate St., Eugene, OR 97403; ph. 503/346-4414.



Look at our Logo list!



Logo users at all levels benefit from these ISTE selections.

The *Introduction to Programming* books, written by Sharon Yoder, provide beginners with a Logo base to build on and experienced users with a reference to rely on. Both are excellent resources for teacher training or introductory computer science classes.

New from ISTE, *LogoWriter (Logo PLUS) for Educators: A Problem Solving Approach* takes Logo learning to new depths. The focus is entirely on learning and practicing general problem solving skills while using Logo. Great for beginning programming experience. Appendices include keystroke summaries, turtle shape pictures, and a quick reference card. Written by Sharon Yoder and Dave Moursund.

To order, contact: ISTE, University of Oregon, 1787 Agate St., Eugene, OR 97403-9905; ph. 503/346-4414.

Telecommunications: Make the connection.

Whether you want to hook up with a teacher in Kenya, or a teacher across town, ISTE's *Telecommunications in the Classroom* will help you make the connection.

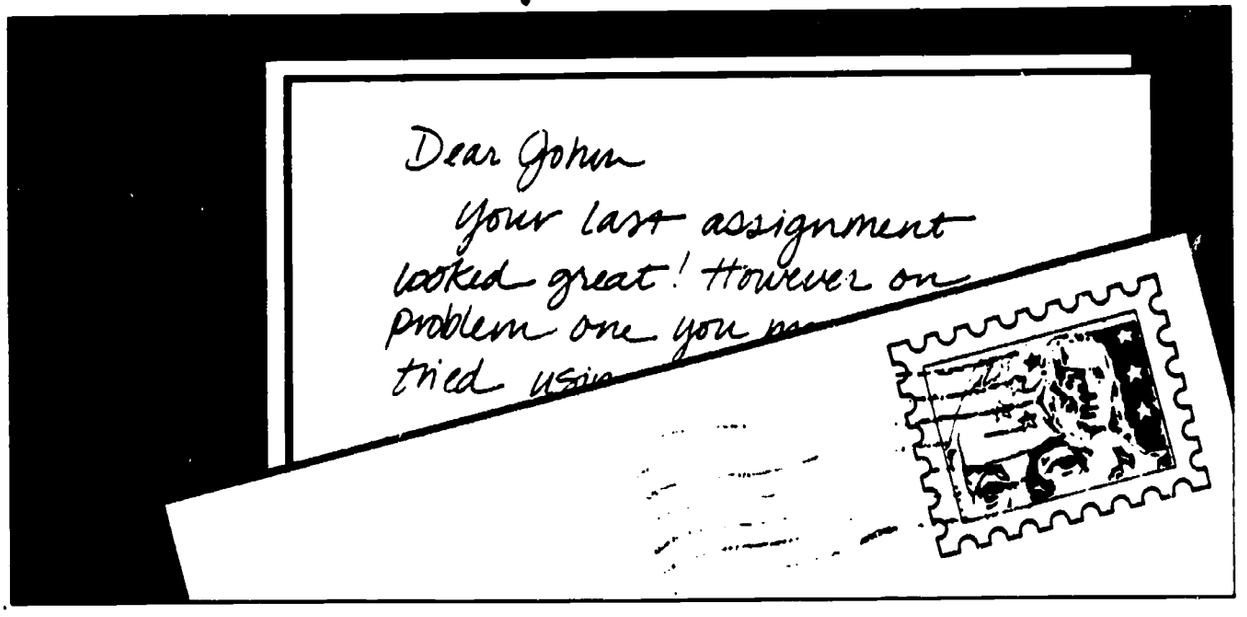
Authors Chris Clark, Barbara Kurshan, Sharon Yoder, and teachers around the world have done your homework in *Telecommunications in the Classroom*. The book details what telecommunications is how to apply it in your classroom, what hardware and software you'll need, and what services are available. *Telecommunications in the Classroom* also includes a glossary of telecommunications terms and exemplary lesson plans from K-12 teachers.

Telecommunications in the Classroom is an affordable, informative resource for workshops, classes, and personal use.

Make your connection today with ISTE's *Telecommunications in the Classroom*

ISTE, University of Oregon, 1787 Agate St., Eugene, OR 97403-9905; ph. 503/346-4414

Finally, a long distance relationship that won't break your heart.



ISTE offers ten *Independent Study* courses that get to the heart of learning.

Each course thoroughly covers the title material and is designed to provide staff development and leadership training. You correspond directly with the course's instructor by mail, and can receive graduate credit through the Oregon State System of Higher Education.

Classes offered this year are:

- Introduction to Logo for Educators (available for LogoWriter or Logo PLUS)
- Fundamentals of Computers in Education
- Long Range Planning for Computers in Schools
- Computers in Mathematics Education
- Computers and Problem Solving
- Introduction to AppleWorks for Educators
- Computers in Composition
- Effective Inservice for Instructional Use of Computers in Education
- Computer Applications for Educators: An Introduction to Microsoft Works
- Telecommunication and Information Access in Education

Register for classes independently or with a group. Districts enrolling six or more teachers receive a fee reduction for each person enrolled.

Courses range in price for 3-4 quarter-hours of graduate credit. You have one year to complete your course.

Start a great long distance relationship today with an ISTE *Independent Study* Course.

Request an *Independent Study* course brochure. Write or call:

ISTE, Independent Study CourseDept.,
University of Oregon, 1787 Agate St.,
Eugene, OR 97403-9905
ph. 503/346-2412

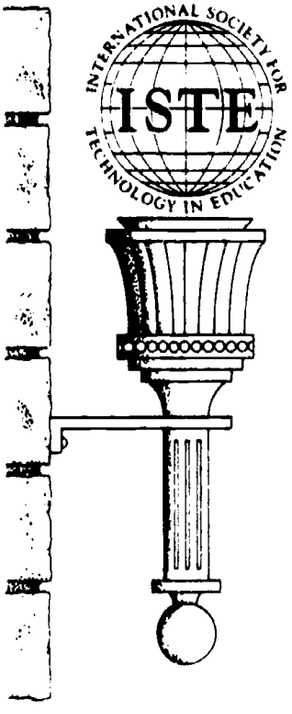
LogoWriter for Educators: A Problem Solving Approach

These innovative texts allow educators to approach programming as an arena for learning and practicing general problem solving skills. Designed as part of a course for inservice/preservice teachers, they provide a solid introduction to Logo and problem solving for any beginning programmer. Carefully sequenced instructions ensure a successful beginning programming experience. New programming ideas are tied to important problem solving concepts. Practical tips include answers to questions frequently asked by beginning Logo programmers, debugging advice to help diagnose and problem-solve programming errors, and suggested activities in both Logo and problem solving. Appendices include summaries of key strokes, pictures of available turtle shapes, and a quick reference card.

LogoWriter for Educators: A Problem Solving Approach
By Sharon Yoder and Dave Moursund
113 pages, 1990
ISBN 0-924667-72-9

Also Available—Logo PLUS for Educators: A Problem Solving Approach

The ***International Society for Technology in Education*** touches all corners of the world. As the largest international non-profit professional organization serving computer using educators, we are dedicated to the improvement of education through the use and integration of technology.



Drawing from the resources of committed professionals worldwide, ISTE provides information that is always up-to-date, compelling, and relevant to your educational responsibilities. Periodicals, books and courseware, *Special Interest Groups*, *Independent Study* courses, professional committees, and the Private Sector Council all strive to help enhance the quality of information you receive.

It's a big world, but with the joint efforts of educators like yourself, ISTE brings it closer. Be a part of the international sharing of educational ideas and technology. Join ISTE.

Basic one year membership includes eight issues each of the *Update* newsletter and *The Computing Teacher*, full voting privileges, and a 10% discount off ISTE books and courseware.

Professional one year membership includes eight issues each of the *Update* newsletter and *The Computing Teacher*, four issues of the *Journal of Research on Computing in Education*, full voting privileges, and a 10% discount off ISTE books and courseware.

Join today, and discover how ISTE puts you in touch with the world.

ISTE, University of Oregon,
1787 Agate St., Eugene, OR 97403-9905.
ph. 503/346-4414